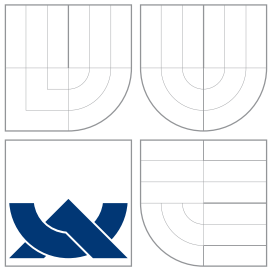
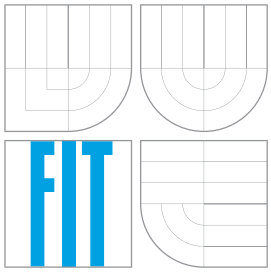


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE SLEDOVÁNÍ SÍŤOVÝCH TOKŮ

OPTIMIZATION OF NETWORK FLOW MONITORING

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. MARTIN ŽÁDNÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2013

Abstrakt

Tato disertační práce se zabývá optimalizací sledování síťových toků. Sledování síťových toků spočívá ve sledování jejich stavu a je klíčovou úlohou pro řadu síťových aplikací. S každým příchodem paketu je nutné aktualizovat hodnoty stavu, což zahrnuje přístupy do paměti. Vzhledem k vysoké propustnosti linek a obrovskému množství souběžných toků hraje přístup do paměti kritickou roli ve výkonnosti stavového zpracování síťového provozu. Tento problém se řeší různými technikami. Tyto techniky ale ve výsledku vždy požadují, aby nejblíže zpracování provozu byla nasazena paměť s nízkou odezvou, cache toků, schopná vyřídit všechny přístupy. Cache toků má proto omezenou kapacitu a její efektivní správa má zásadní vliv na výkonnost a výsledky zpracování síťového provozu. Vzhledem ke specifikům síťového provozu nemusí být stávající správy vhodné pro správu cache toků. Disertační práce se proto zabývá automatizovaným vývojem správy cache na základě reálného provozu dané sítě. Automatizace vývoje správy cache toků je realizována pomocí genetického algoritmu. Genetický algoritmus vyvíjí nová řešení a hodnotí je simulací nad vzorkem provozu z různých sítí. Navržený postup je ověřen na vývoji správ pro dva problémy. Prvním problémem je vývoj správy, která bude vykazovat celkově nízký počet výpadků stavů z cache toků. Druhým problémem je vývoj správy, která bude vykazovat velmi nízký počet výpadků u velkých toků. Optimalizace zakódování správy a experimenty s parametry genetického algoritmu ukázují, že je možné nalézt správy cache toků, které jsou optimalizované pro specifika daného nasazení. Nově vyvinuté správy poskytují lepší výsledky než ostatní testované správy. Z hlediska snížení celkového počtu výpadků je vyvinuta správa, která snižuje počet výpadků na konkrétní datové sadě až o deset procent vůči nejlepší porovnávané správě. Z pohledu snížení počtu výpadků u velkých toků je dosaženo vyvinutou správou až dvojnásobného snížení výpadků. Většina velkých toků (více než 90%) nezaznamenala při použití vyvinuté správy dokonce ani jeden výpadek. Rovněž během záplav nových toků, které se v síťovém provozu vyskytují v souvislosti se skenováním sítí a útoky, se ukazují velmi dobré vlastnosti vyvinuté správy. V rámci práce je rovněž navrženo rozšíření správy o využití doplňkové informace ze záhlaví příchozích paketů. Výsledky ukazují, že kombinací této informace lze počet výpadků u správ dále snižovat.

Klíčová slova

Síť, IP tok, genetický algoritmus, správa paměti, simulace.

Citace

Martin Žádník: Optimalizace sledování síťových toků, disertační práce, FIT VUT v Brně, Brno, 2013

Abstract

This thesis deals with optimization of network flow monitoring. Flow-based network traffic processing, that is, processing packets based on some state information associated to the flows which the packets belong to, is a key enabler for a variety of network services and applications. The number of simultaneous flows increases with the growing number of new services and applications. It has become a challenge to keep a state per each flow in a network device processing high speed traffic. A flow table, a structure with flow states, must be stored in a memory hierarchy. The memory closest to the processing is known as a flow cache. Flow cache management plays an important role in terms of its effective utilization, which affects the performance of the whole system. This thesis focuses on an automated design of cache replacement policy optimized to a deployment on particular networks. A genetic algorithm is proposed to automate this process. The genetic algorithm generates and evaluates evolved replacement policies by a simulation on obtained traffic traces. The proposed algorithm is evaluated by designing replacement policies for two variations of the cache management problem. The first variation is an evolution of the replacement policy with an overall low number of state evictions from the flow cache. The second variation represents an evolution of the replacement policy with a low number of evictions belonging to large flows only. Optimized replacement policies for both variations are found while experimenting with various encoding of the replacement policy and genetic operators. The newly evolved replacement policies achieve better results than other tested policies. The evolved replacement policy lowers the overall amount of evictions by ten percent in comparison with the best compared policy. The evolved replacement policy focusing on large flows lowers the amount of their evictions two times. Moreover, no eviction occurs for most of the large flows (over 90%). The evolved replacement policy offers better resilience against flooding the flow cache with large amount of short flows which are typical side effects of scanning or distributed denial of service activities. An extension of the replacement policy is also proposed. The extension complements the replacement policy with an additional information extracted from packet headers. The results show further decrease in the number of evictions when the extension is used.

Keywords

Network, IP flow, genetic algorithm, replacement policy, simulation.

Optimalizace sledování síťových toků

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením Prof. Ing. Lukáše Sekaniny, Ph.D. a školitele specialisty Ing. Jana Kořenka, Ph.D.

.....
Martin Žádník
6. února 2013

Poděkování

Chtěl bych poděkovat Prof. Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení mé disertační práce a školiteli specialistovi Ing. Janu Kořenkovi, Ph.D. za cenné rady a připomínky.

Obsah

1 Úvod **3**

2 Sledování toku **5**
2.1 Definice toku 5
2.2 Přenos dat a sledování stavu toku 6
2.3 Směrování na úrovni toků 7
2.4 Měření toků 8
2.5 Shrnutí a motivace výzkumu 9
2.6 Cíle práce 11

3 Vlastnosti síťového provozu **12**
3.1 Soběpodobnost a rozložení vlasností síťového provozu 12
3.2 Složení provozu a sledované charakteristiky 13
3.3 Třídy toků a jejich vlastnosti 13
3.4 Použité datové sady a jejich charakteristiky 15
3.5 Úprava definice sloních toků 18

4 Správy paměti cache **22**
4.1 Triviální správy paměti 22
4.2 Správa paměti využívající nedávnou historii přístupů 23
4.3 Správa cache využívající četnost přístupů 25
4.4 Adaptivní správa cache 26
4.5 Genetické algoritmy a cache 27
4.6 Optimální správa cache 28
4.7 Filtry 28
4.8 Zhodnocení 30

5 Návrh správy cache toků **32**
5.1 Definice problému 32
5.2 Reprezentace správy cache 33
5.3 Struktura cache toků 35
5.4 Návrh správy cache toků pomocí genetického algoritmu 37
5.4.1 Fitnes funkce 38
5.4.2 Zakódování úlohy správy cache 40
5.4.3 Genetické operátory 45
5.5 Zkrácení doby ohodnocení 47

6 Experimenty **51**
6.1 Vliv optimalizací a genetických operátorů 51
6.2 Vývoj správy cache velkých toků 57
6.3 Porovnání s ostatními správami 59
6.4 Adaptace správy cache 64

7 Správa paměti ve světle útoků **68**
7.1 Rozšíření datové sady o anomálie 69
7.2 Hodnocení správ 70

8 Rozšíření správy **76**
8.1 Příznaky a vzdálenost následujícího paketu 77
8.2 Příznaky a velké toky 83
8.3 Návrh klasifikátoru 87
8.4 Kombinace výstupu klasifikátoru a správy cache 90
8.5 Výsledky 92

9 Systém pro organizaci síťového provozu **95**
9.1 Motivace a výzvy 95
9.2 Návrh systému 96
9.3 Implementace 97
9.3.1 Vyhledání v cache 98
9.3.2 Správa cache a implementace v FPGA 99
9.4 Hodnocení systému 100

10 Závěr **103**
10.1 Přínosy práce 104
10.2 Budoucí práce 104

11 Příloha **106**
11.1 Výpis posloupností 106
11.2 Graficky zobrazené správy cache 108
11.3 Výsledky hledaní bodu vložení pro SLRU 109
11.4 Nastavení klasifikátoru vzdáleností 110
11.5 Vyvinuté správy cache 113

Kapitola 1

Úvod

Objem síťového provozu a přenosová šířka síťových linek stále roste, stejně tak roste i množství uživatelů a služeb poskytovaných přes Internet. Neustálý růst klade stále vyšší nároky na síťová zařízení. Typickými požadavky jsou především vysoká propustnost a robusnost, které jsou odrazem požadavku na vysokou kvalitu a dostupnost služeb. Zvyšování propustnosti vede ke snižování času, který je dostupný pro zpracování dat na síťových zařízeních. Tato zařízení vykonávají nad síťovým provozem rozličné operace a zajišťují nezbytné služby pro správné fungování sítě. Mezi příklady nejčastěji se vyskytujících síťových zařízení patří směrovače, filtrovací zařízení (*firewally*), aplikačně-specifické brány (*proxy*), systémy pro detekci útoků a průniků, sondy pro sběr a měření síťového provozu a další.

Přenos dat mezi koncovými uzly v paketově orientovaných sítích se děje rozdělením dat do malých bloků, které jsou ve formě paketů přenášeny přes síť. V koncovém uzlu jsou data z paketů opět spojována do souvislého celku. Stav přenosu proudu dat je sledován nejen ve zdrojovém a cílovém uzlu, ale i v převážné většině zařízení v síti. Sledování stavů síťových toků umožňuje síťovým zařízením vykonávat základní ale i komplexní operace nad procházejícími daty. Sledování toků lze nalézt ve směrovačích, kde sledování slouží pro dodržení kvality služby, pro překlad[1] IP adres a k urychlení samotného směrování. Filtrovací zařízení sledují stavy toků, aby byly schopné zabránit neoprávněným přístupům do chráněné sítě nebo naopak propustit povolenou komunikaci. Rovněž systémy pro detekci útoků a škodlivého provozu sledují stavy toků, aby byly schopny odhalit škodlivý provoz (například podezřelé řetězce rozdělené do více paketů). V neposlední řadě jsou toky sledovány pomocí síťových sond a výsledky tohoto sledování jsou nepostradatelnou součástí při správě, plánování a ochraně počítačové sítě.

Samotné sledování stavu několika toků je jednoduchá, výpočetně nenáročná činnost. V síťových zařízeních se ale sledují stavy toků pro všechna současně probíhající spojení. Díky rostoucímu počtu uživatelů a služeb je počet současně aktivních toků na síti velmi vysoký. Stavové zpracování je navíc specifické tím, že s příchodem paketu je nutné získat stav příslušného toku z paměti, tento stav aktualizovat a opět do paměti uložit. Sledování toků na linkách s vyšší rychlostí představuje výzvu současným technologiím, neboť použitá paměť musí mít vysokou propustnost, nízkou latenci a zároveň pojmout velké množství dat. Takové požadavky lze považovat za protichůdné a paměť splňující všechny tyto podmínky by byla velmi drahá nebo by ji nebylo možné vyrobit.

Nejrozšířenějším způsobem řešení protichůdných požadavků na paměť je uspořádání několika typů pamětí do hierarchické struktury. Paměť na nejnižší úrovni, nejblíže sa-

[1] Překlad IP adres je znám spíše pod anglickým názvem Network Address Translation (NAT)

motnému zpracování, musí být dostatečně rychlá, aby s příchodem každého paketu poskytla a uložila potřebné informace. Její velikost je ale omezena. Tato paměť je v oblasti sledování stavů toků označována jako cache[2] toků (z anglického termínu *flow cache*). Správa této cache hraje důležitou roli z pohledu jejího efektivního využití a nasazení v konkrétní aplikaci.

Přestože v současné době existuje široká paleta správ cache, není zcela zřejmé, jaká správa bude pro dané nasazení v dané síťové aplikaci nejlepší. Dále zůstává otázkou, zda je možné pro dané nasazení nalézt lepší správu paměti než jsou existující přístupy. Prostor pro zlepšení vzniká optimalizací správy paměti na specifické vlastností vstupních dat, požadavky dané aplikace a velikost použité cache.

Nejlepších výsledků při správě cache toků dosahuje algoritmus pracující tzv. offline. Tento algoritmus využívá znalost o budoucím vývoji zastoupení položek v přicházejících datech pro rozhodnutí o uvolnění položek z cache. Bylo dokázáno, že při různé velikosti položek v cache je nalezení optimální offline správy cache NP-těžký problém [17]. Důkaz byl veden pomocí redukce na KNAPSACK problém [42]. Cache je vnímána jako kolekce položek, kdy položka má definovanou svou váhu jako velikost paměťového místa zabraného položkou a svou hodnotu definovanou jako počet přístupů k položce. Stejně jako u KNAPSACK problému je cílem vybrat a umístit do cache takové položky, aby byla maximalizována celková suma hodnot položek v cache a zároveň nebyla přesažena kapacita cache.

Reálné správy (bez znalosti budoucích přístupů) pracují nad proudem dat a ze své podstaty jsou vždy založeny na heuristice. Heuristika může pracovat pouze s daty, která již daným zařízením prošla v minulosti. Na základě této znalosti se heuristika snaží přiblížit výsledkům optimálního řešení offline správy cache.

Bylo navrženo mnoho heuristik pro správu cache v různých oblastech (architektura procesorů, cache doménových jmne, cache objektů v Internetu apod). Tyto heuristiky byly navrhovány na základě zkušenosti návrháře znalého složení konkrétních dat. Charakteristiky provozu na různých typech linek mohou být ovšem rozdílné. Navíc postihnout všechny tyto charakteristiky při návrhu správy cache toků může být velmi obtížné. V neposlední řadě se charakteristiky síťového provozu mohou značně lišit od typického dosavadního nasazení správy cache v oblasti architektury počítačů a procesorů. Proto bude v této práci zaveden nový přístup pro návrh správy cache toků. Tento přístup bude umožňovat automatizovaně vyvinout správu cache toků respektující charakteristiky provozu určité sítě. Za účelem vývoje správy cache toků bude využit genetický algoritmus. Genetický algoritmus se ukázal již dříve jako dobrý prostředek pro hledání nových řešení v podobných typech úloh.

Tato práce je členěna následovně. Druhá kapitola se zabývá popisem a aplikacemi sledování stavů toků. Následující kapitola pojednává o vlastnostech síťového provozu a popisuje vlasnosti zvolených datových sad. Ve čtvrté kapitole jsou diskutovány doposud navržené přístupy správy cache. Pátá kapitola je věnována návrhu správy cache toků. Zároveň jsou navržený různé optimalizace, které umožní rychlý vývoj vhodné správy cache. Následně jsou provedeny experimenty a porovnány výsledky různých typů správ. Kapitola 7 analyzuje správu cache ve světle útoků založených na záplavě toků. V kapitole 8 je navrženo rozšíření správy cache založené na detekci příznaků ze záhlaví paketu. V deváté kapitole je popsán systém pro organizaci síťového provozu využívající specifickou správu cache. Závěry práce jsou diskutovány v desáté kapitole.

[2]Výraz cache se běžně využívá pro označení rychlé vyrovnávací paměti v procesorech

Kapitola 2

Sledování toku

2.1 Definice toku

Pro potřeby této práce je využita definice síťového toku v IP síti dle RFC 5101 [20], které představuje nejnovější standard pro sledování a přenos záznamů o síťových tocích pomocí protokolu IPFIX (IP Flow Information Export). RFC 5101 definuje tok jako sekvenci paketů, která je za určitý časový úsek přenesena přes místo pozorování. Pakety daného toku mají společnou vlastnost, která je funkcí údajů v záhlaví paketu, či charakteristikou paketu samotného, například jeho délka. V krajním případě lze za jeden tok považovat všechny pakety v provozu, pokud funkce náležitosti paketu vrací pro jakýkoliv paket vždy stejný výsledek. V opačném případě může být jako jeden tok vnímán každý paket, pokud funkce pro každý příchozí paket vrátí různý výsledek, například pořadové číslo ze záhlaví protokolu TCP nebo čas příchodu paketu na síťové rozhraní.

V závislosti na aplikaci se definice toku zúží. Každá aplikace může identifikovat tok na základě rozdílných parametrů. Například při směrování paketů může být výhodné tok definovat pouze na základě cílové IP adresy. Nejčastěji je ale využívána definice toku jako množiny paketů, která má shodnou pětici údajů ve svém záhlaví. Touto pěticí jsou:

- zdrojová a cílová IP adresa,
- zdrojový a cílový port,
- číslo protokolu.

V této práci je využívána definice toku jako sekvence paketů se shodnou, výše uvedenou, pěticí údajů. Důvodem oblíbenosti této definice je možnost rozlišení provozu až na úroveň dvou komunikujících programů na síťových entitách.

Tok je prohlášen za ukončený po uplynutí intervalu neaktivity (nepřišel žádný paket náležící danému toku). Tento interval se může lišit dle aplikace (většinou od 30 s do 5 i více minut). Z hlediska správy cache ale tento interval nehraje roli. Správa cache toků je spuštěna v okamžiku, kdy je cache plná a je nutné uvolnit místo pro nový tok. Není tedy možné čekat po vymezený interval na neaktivitu toku. V některých případech může být konec toku rozpoznán na základě nastavených příznaků, například u toků obsahující protokol TCP.

Sledování stavu toku znamená sběr a správu informací, které identifikují daný tok a popisují jeho průběžný stav a vlastnosti. Tato činnost v sobě zahrnuje několik konkrétních úloh a to především příjem paketů na síťovém rozhraní, výpočet funkce náležitosti paketu k toku, vyhledání záznamu o daném toku, aktualizace a uložení informace o paketu do

záznamu. Následně je nutné rozpoznat ukončení toku, aby bylo možné zabranou paměť využít pro nově vznikající toky.

Sběr informace o toku lze popsat formálně. Síťový provoz v určitém intervalu lze vnímat jako uspořádanou množinu paketů $P = \{\rho_1, \rho_2, \ldots, \rho_{|P|}\}$, kde uspořádání je dáno pořadím příchodů paketů na síťové rozhraní. Množinu P lze rozdělit na základě příslušnosti k tokům do vzájemně disjunktních uspořádaných podmnožin $f_1, f_2, \ldots, f_{|F|}$ tvořících množinu toků F ($|F|$ je počet toků v P):

$$\forall i, j, 1 \leq i \leq |F|, 1 \leq j \leq |F|, i \neq j : f_i \cap f_j = \emptyset,$$

kde relace uspořádání na množině f_i je definována na základě uspořádání paketů v P.

Funkce h je bijektivní zobrazení $h : P \rightarrow F$ přiřazující každému paketu $\rho \in P$ tok $f \in F$. Při příchodu paketu ρ_k je vyhledán příslušný stav toku $st(h(\rho_k))$ a je aktualizován aplikací funkce ak. Potom:

$$st(h(\rho_{k+1})) = ak(\rho_k, st(h(\rho_k))), k \ = 1, 2, \ldots, \mid P \mid .$$

Je vidět, že pro běžné sledování stavu toku f_i není potřeba informace (pakety) o toku jiném. Z toho vyplývá, že pakety různých toků je možné vzájemně libovolně prokládat při zachování uspořádání v rámci toku, aniž by byl ovlivněn výsledek. Pokud pořadí není zachováno, pak lze v některých případech obnovit původní pořadí (například u TCP). Nicméně cena obnovy pořadí paketů v toku je poměrně vysoká, neboť vyžaduje paměť pro uložení celých paketů, které přijdou mimo pořadí. V některých případech je aktualizační funkce závislá i na dalších parametrech, které nelze získat z příchozího paketu nebo stávajícího stavu toku. Například při sledování množství paketů, které přišly na síťové rozhraní mezi příchody dvou nejbližších paketů daného toku. Pak musí být dodrženo pořadí příchodů paketů dle pořadí v sekvenci P.

Nyní zavedeme konvenci pro popis některých význačných veličin toku. Velikost toku f_i je množství paketů, které tok přenáší a velikost je značena $|f_i|$. Objem toku je počet přenesených bajtů a bude značen $|f_i|_b$. Trvání $|f_i|_t$ toku f_i je velikost časového úseku mezi prvním paketem toku a posledním paketem toku.

2.2 Přenos dat a sledování stavu toku

Přenos dat mezi koncovými uzly v paketově orientovaných sítích děje rozdělením přenášených dat do více menších bloků (minimální a maximální velikost bloku je dána použitým protokolem a linkou). Následně jsou tyto bloky přeneseny přes síť za použití různých protokolů. Na koncovém uzlu jsou data opět spojována do souvislého celku. V sítích založených na protokolu IP je pro správné zpracování nutné stav toku dat sledovat, aby bylo možné řídit přenos a správně spojovat příchozí bloky dat. Sledování toku se děje v souladu s popisem toku jako sekvence paketů se shodnou pětic údajů (zdrojových a cílových IP adres, portů, a čísla protokolu).

V současné době je nejčastěji využívaným protokolem aplikační vrstvy protokol TCP a následně UDP. U protokolu TCP se spojení sestává ze dvou toků opačného směru. Sledování TCP spojení vyžaduje uchování záznamu zvaného TCB (TCP/Transmission Control Block). Tento záznam obsahuje informace o stavu spojení, odhad aktuálního RTT (Round Trip Time), informace pro řízení zahlcení a údaje pro zpracování dat (ukazatele na různé fronty dat, velikost okna, čítače). V případě protokolu UDP je množství informací, které

jsou potřebné pro sledování stavu, menší, neboť není nutné udržovat informace o zahlcení a řízení nebo nepodařených přenosech. Přesto velikost záznamu není zanedbatelná. V nejmenší podobě je nutné udržovat identifikátor toku (tj. již dříve představenou pětici údajů), která dohromady zabírá 13 B, pokud budeme uvažovat pouze IPv4 adresy. V dnešní době je však nezbytné uvažovat i IPv6 a pak identifikátor toku bude zabírat 37 B. Na velikosti záznamu se dále podílí množství informací, které se k danému toku udržují. V případě sledování TCP je velikost TCB 512 B. Vezmeme-li v úvahu fakt, že využívání protokolu TCP má značnou převahu nad UDP, pak nutně pamětové nároky zařízení sledující datové toky za účelem rekonstrukce dat z paketů musí mít k dispozici značné množství paměti pro uložení všech údajů o souběžných tocích.

Zpracování dat přenášených protokolem TCP je implementováno především v koncových uzlech na síti. V případě, že koncovým uzlem je server poskytující služby, pak sledování stavů toků může být kritická úloha, neboť na serveru se sbíhá mnoho souběžných spojení. K vytíženým zařízením se proto dodává *TCP Offload Engine* (TOE) [44], který bývá implementován ve specializované síťové kartě. Princip TOE spočívá v přesunu zpracování síťové a transportní vrstvy přímo na kartu, která tak sleduje stavy síťových toků a skládá přijímaná data do větších celků, které jsou přenášeny do paměti hostitelského počítače. TOE tak šetří paměť a procesorový čas serveru.

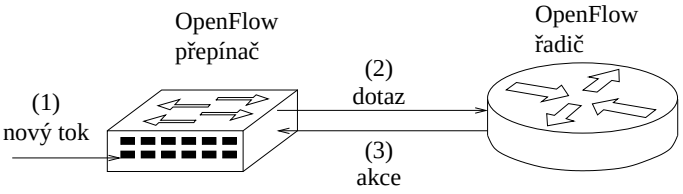
Kromě koncových systémů je mnohdy potřeba sledovat TCP spojení i v síti, například v aplikačních branách (*proxy*), které mimo jiné mohou řídit a filtrovat datový obsah provozu, zajišťovat šifrování a další služby. Proxy pracuje tak, že na jedné straně přijímá TCP spojení a na druhé straně další spojení vytváří. V proxy pak dochází k rekonstrukci dat z prvního spojení, jejich zpracování a následně k odeslání zpracovaných dat pomocí druhého spojení.

Obsah dat je rovněž předmětem zájmu systému pro dekekci průniků (Intrusion Detection System, zkráceně IDS). Tyto systémy vyhledávají v datech škodlivé vzory, například vložení škodlivého dotazu do formuláře (SQL injection). V omezené míře implementují sledování TCP spojení i stavové firewally, které si při vytvoření nového TCP spojení založí stav. Tento stav umožní propustit přidružený tok z opačného směru spojení, přestože by byl jinak zakázán. Následně je možné sledovat například sekvenční číslo v TCP datagramu a zahodit ty, u nichž je sekvenční číslo příliš vzdáleno od očekávaného.

2.3 Směrování na úrovni toků

Sledování toků se využívá nejen k rekonstrukci a řízení proudu dat, ale často slouží pro urychlení síťových operací jako je směrování a klasifikace paketů. Princip urychlení spočívá v převodu problému směrování (například vyhledání cesty na základě nejdelšího shodného prefixu) nebo klasifikace každého paketu na základě sady pravidel na problém vyhledání stavu toku. Převod na problém vyhledání stavu toku je výhodný v situaci, kdy výpočetní náročnost původního problému je větší než počet současně aktivních toků. Při příchodu prvního paketu každého toku se jednou vyhodnotí akce, která se pak vykonává nad dalšími pakety daného toku.

Jako příklad lze uvést OpenFlow architekturu [15]. Ta je založena na dvou typech prvků, OpenFlow přepínači a OpenFlow řadiči. Přepínač si udržuje informace o akcích, které má vykonat nad průchozími toky, například zda tok povolit, kam ho dále přeposlat apod. Pokud o daném toku nemá dostupné informace, zašle dotaz řadiči. Řadič zpracuje dotaz, vybere správnou akci a zašle odpověď na přepínač, který si uloží danou informaci v podobě záznamu o toku. Toky jsou v OpenFlow definovány opět na základě pětice údajů, což



Obrázek 2.1: Architektura OpenFlow.

dovoluje každé aplikaci přiřadit jinou akci a ovlivnit způsob průchodu sítí. Tento systém je zobrazen na obr. 2.1.

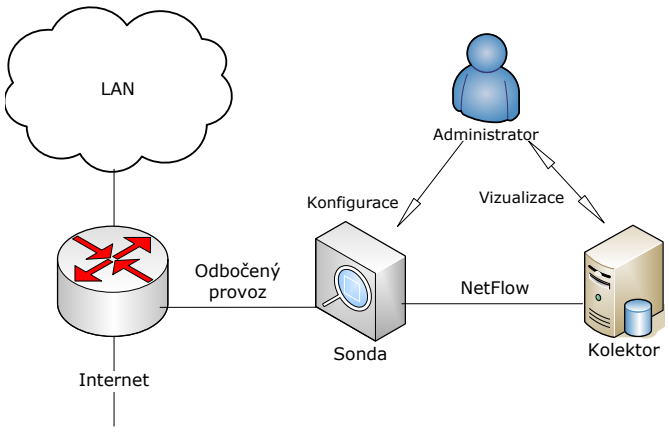
Podobně mohou směrovače přesouvat výsledky vyhledání ze značně rozsáhlé směrovací tabulky do menší a levnější směrovací paměti. Na rozdíl od OpenFlow přepínače se využívá jiné definice toku. Jako identifikátoru toku je využito části cílové IP adresy a jako stav se uchovává IP adresa následujícího skoku (next hop address). Nicméně mnohé směrovače vykonávají další činnosti kromě směrování. Tyto činnosti vyžadují rovněž sledování toků, v mnoha případech s definicí na základě pětice IP adres, portů a protokolu. Jednou z těchto činností je rozpoznání aplikace (Cisco NBAR), která využívá daný tok k přenosu dat. Na základě této informace lze řídit přenosovou šířku pásma pro dané toky (tzv. traffic-shaping) nebo jejich specifické směrování. Dále mohou směrovače překládat adresy paketů při směrování mezi různými sítěmi, tzv. NAT. NAT v okamžiku příchodu prvního paketu toku přepíše zdrojovou IP adresu paketu na výstupní adresu routeru a vygeneruje nové číslo zdrojového portu. O tomto překladu je založen záznam (stav toku), který dovoluje překládat další pakety toku ve shodě s překladem prvního paketu.

2.4 Měření toků

Sledování síťového provozu je nepostradatelnou součástí při správě, plánování a ochraně počítačové sítě. Dnešní sítě mají složitou a rozsáhlou topologii (fyzickou i logickou). Bez sledovací infrastruktury je ověření správného nastavení síťových prvků nebo řešení incidentů, jako je zvýšení chybovosti linky, špatné směrování a dalších patologií, jen obtížně proveditelné. Navíc spektrum současných aplikací, jako jsou IP telefonie, video konference, burzovní a bankovní transakce, klade vysoké nároky na spolehlivost, kvalitu a bezpečnost komunikace, což jen podtrhuje význam sledování sítě.

Sledování provozu na úrovni toků se ukazuje jako vhodný typ měření, jenž bude možné využít i v budoucnu. Jeho výhodou je především nezávislost na obsahu paketu a je tudíž odolná vůči pokusům obejít měření za pomoci šifrování, které je dnes u většiny moderních aplikací běžné. Navíc neporušuje soukromí komunikace uživatelů, tedy neprohledává obsah komunikace, ale vystačí si pouze se záhlavím paketu. V neposlední řadě jsou naměřené výsledky široce využitelné a lze je uplatnit i ve velké většině případů spojených se správou, plánováním a ochranou sítě. Jako příklad lze uvést měření kvality spojení s využitím statistik o mezi-paketových intervalech, kdy u spojení přenášejících hlas lze očekávat, že interval mezi dvěma následujícími pakety nepřesáhne určitou dobu (obvykle 20 ms). Toto měření poskytuje informaci o zdroji a cíli komunikace, o komunikující aplikaci, o času a délce trvání komunikace, o množství přenesených dat a podobných statistikách.

Síťové toky jsou velmi často předmětem měření, neboť poskytují vhodný pohled na probíhající komunikaci. Oproti sběru paketů je sbíráno daleko méně dat, nicméně jejich informační hodnota je dostačující pro řešení většiny problémů, které jsou spojeny se správou



Obrázek 2.2: Architektura měření a sběru NetFlow dat.

sítě. Data o tocích umožňují daleko lépe dohledávat incidenty na síti, ukázat vytížení sítě v různých časových obdobích nebo ladit nastavení kvality služby.

Mezi široce používané technologie pro sledování toků v síti patří NetFlow [1], která byla představena firmou Cisco koncem devadesátých let. Popularita NetFlow plyne z masového rozšíření Cisco routerů, které NetFlow podporují. Časem se NetFlow stalo *de facto* standardem pro získávání dat o tocích ze sítě a bylo vypracováno i informační RFC [1] 3954 [19] o protokolu NetFlow v9.

Infrastruktura NetFlow (obr. 2.2) je postavena na dvou typech prvků. Prvním typem jsou sondy schopné sbírat statistiky (záznamy o tocích) a odesílat je pomocí NetFlow protokolu. Druhým typem prvku je kolektor, který přijímá naměřené statistiky a ukládá je pro další analýzu. Sondy mohou být implementovány v routerech nebo autonomních sondách, které jsou umístěny do důležitých míst sítě. Například do místa připojení lokální sítě do Internetu, do uzlů kampusových sítí nebo do datových center. Kolektor může být umístěn kdekoliv v síti a sbírat informace z několika sond zároveň. Záznamy z těchto měření se dlouhodobě uchovávají například pro zpětné dohledání reportovaných incidentů.

Efektivní správa cache toků v sondě hraje klíčovou roli. Cílem je udržet co nejvíce toků po celou dobu jejich existence tak, aby nedošlo k odstranění toku stavu z cache před skutečným koncem toku. V takovém případě vzniká více záznamů a snižuje se tak míra agregace příchozích paketů a toků. Tento stav ztěžuje následnou analýzu těchto dat a zvyšuje nároky na výpočetní zdroje kolektoru.

2.5 Shrnutí a motivace výzkumu

Sledování toků podporuje nebo přímo zajišťuje některé významné operace na počítačové síti jako jsou přenos dat, směrování, měření síťového provozu a další. Sledování několika toků je jednoduchá a nenáročná činnost, nicméně sledování všech souběžných toků na páteřních linkách je velmi náročné z pohledu rychlosti a velikosti paměti.

Sledování stavů toků vyžaduje s každým příchodem paketu vykonat následující operace:

[1] Request for Comment

Tabulka 2.1: Interval mezi příchody nejkratších paketů.

Rychlost [Gbps]	10	40	100
Max. počet paketů za sekundu	$1,49 \cdot 10^7$	$5,9 \cdot 10^7$	$1,49 \cdot 10^8$
Doba pro zpracování paketu [ns]	67	17	7

příjem paketů na fyzickém rozhraní, nalezení relevantních polí v záhlaví paketu, výpočet funkce náležitosti paketu k toku, vyhledání stavu toku, jeho aktualizace informací z paketu a uložení zpět do paměti.

Kritické místo v tomto zpracování představuje přístup do paměti za účelem vyhledání a aktualizací stavu toku (ostatní operace lze dobře paralelizovat). Časový limit pro tuto operaci je dán rychlostí příchodů paketů na dané lince. Tabulka 2.1 zobrazuje časy, které jsou v případě paketů délky 64 B k dispozici pro zatřídění informace z paketu do stavu toku.

Kromě rychlosti linek vzrůstá počet uživatelů a aplikací, čímž se neustále zvyšuje počet toků [3]. Na linkách s vysokou přenosovou rychlostí a s velkým množstvím uživatelů je velmi obtížné sledovat stav všech současně existujících toků. Použitá paměť by musela mít dostatečně krátkou dobu čtení a zápisu, aby bylo možné provést aktualizaci stavu toku s příchodem každého paketu, a zároveň by musela být dostatečně velká, aby bylo možné uchovat stavy všech současně aktivních toků. Tyto dva požadavky jsou ovšem protichůdné a paměť splňující oba požadavky zároveň je velmi drahá, anebo ji není možné se současnými technologiemi vyrobit.

Z tohoto důvodu jsou vytvářeny víceúrovňové architektury pamětí. Paměť v první úrovni má dostatečnou propustnost a dobu odezvy pro zpracování přicházejících dat, ale má malou kapacitu. Proto je doplněna o paměť v druhé úrovni, která kompenzuje nedostatečnou kapacitu paměti v první úrovni. Ostatní parametry paměti v druhé úrovni mohou být horší, neboť jsou kompenzovány pamětí v první úrovni. Taková architektura může dobře pracovat za předpokladu, že většina přístupů je vyřízena pamětí v první úrovni. Proto správa této paměti hraje významnou roli ve výkonnosti celého systému a v kvalitě poskytovaných dat.

Využití nebo nevyužití více úrovní záleží na požadavcích dané aplikace. Aplikace, které vyžadují souvislé sledování stavu toku od jeho prvního paketu až po poslední paket, nutně potřebují více úrovní paměti, které dovolí odsunout stav toku z paměti na nižší úrovni a uchovat ho v paměti na úrovni vyšší. Některé aplikace ale mohou tolerovat úplné odstranění stavu toku z paměti i po dobu, kdy je tok stále aktivní, a nepotřebují tak složitou hierarchii pamětí. Po odstranění a příchodu paketu, který náleží danému toku, je stav znovu vytvořen. Příkladem tohoto druhu aplikací může být měření síťových toků. Při odstranění aktivního nebo neaktivního toku z paměti je výsledek reportován na kolektor. Pokud tok dále pokračuje, pak musí být založen nový stav, a tok je dále souvisle sledován do dalšího odtranění. Z jednoho toku tak vznikne několik menších. Přestože tento stav není žádoucí, neboť ztěžuje zpracování a analýzu nasbíraných dat, lze ho do určité míry tolerovat. V případě směrování pomocí OpenFlow je možné stav odstraněného toku získat opětovným dotazem na kontrolér. Je jasné, že čím více dotazů, tím vyšší režie (ve smyslu množství vyrovnávací paměti na přepínači pro zachycení provozu, který není možné směrovat) a tím vyšší vytížení kontroleru. V jiných situacích lze stav toku po jeho odstranění obnovit výpočtem (směrovací tabulka směrovače). Pro potřeby některých aplikací dokonce postačí zaměřit se pouze na sledování stavů nejvýznamnějších toků z pohledu množství přenesených dat. Těchto toků je řádově méně než všech toků, nicméně mají největší podíl na objemu přenesených dat (v

anglické literatuře se tyto toky označují jako velké/těžké případně sloní toky). Pro některé linky platí, že deset procent toků přenáší více jak devadesát procent objemu dat.

Paměť uchovávající stavy toků na nejnižší úrovni hierarchie, tj. nejblíže samotnému zpracování síťového provozu, nazýváme pak cache toků. Správa této cache představuje kritickou úlohu z pohledu rychlosti a kvality výsledků ve všech případech použití. Nalezení efektivní správy pro potřeby stavového zpracování síťového provozu je předmětem zkoumání v rámci následujících kapitol této práce.

2.6 Cíle práce

Cílem této disertační práce je výzkum automatizovaného návrhu správy cache, především správy cache toků, na základě vstupních dat, požadavků cílové aplikace a konkrétní velikosti a struktury cache. Rovněž bude experimentálně ověřena hypotéza, zda výsledná správa může dosáhnout lepších výsledků než doposud navržená řešení. Základní úlohou správy je snížit počet výpadků při vyhledání stavů toků v cache. Ve světle složení síťového provozu a různorodosti síťových zařízení bude pozornost zaměřena nejen na základní úlohu správy ale i na její variace. Tyto variace spočívají v optimalizaci správy pro určitou množinu toků. Dalším cílem je tedy ověřit důsledky optimalizací, a to konkrétně, zda lze vyvinout správu zaměřující se na toky s velkým objemem dat a je díky tomu možné snížit paměťové nároky při sledování stavů toků nebo zvýšit odolnost zařízení proti některým síťovým útokům. Dalším cílem je zjistit, zda je možné správu rozšířit o další informaci ze záhlaví příchozích paketů a následně porovnat rozšířenou správu s ostatními. V neposlední řadě je cílem práce ukázat, že navržený přístup lze implementovat v reálném zařízení, které zpracovává síťový provoz na vysokých rychlostech.

Kapitola 3

Vlastnosti síťového provozu

Síťový provoz je specifický svým složením a vlastnostmi, které ovliňují úspěšnost správy cache. Velmi důležitými faktory je rozložení velikosti toků co do počtu paketů, délky trvání toků nebo tvorby shluků. Nerovnoměrné zastoupení, odchylky a neuniformní rozložení těchto vlastností umožňuje optimalizaci správy pro síťový provoz. Tato kapitola popisuje obecně známé a zkoumané vlasnosti síťového provozu. Dále obsahuje popis a rozbor datových sad používaných v této práci.

3.1 Soběpodobnost a rozložení vlasností síťového provozu

Představa o síťovém provozu se v průběhu minulých let vyvíjela. Zpočátku panovalo přesvědčení, že síťový provoz je podobný klasickému telefonnímu a tudíž, že většina jeho vlastností se řídí Poissonovým rozložením [50]. Empirické studie reálných vzorků dat ale ukázaly, že síťový provoz je tzv. soběpodobný ve velkém rozsahu časových měřítek [69], což odporuje teorii o Poissonovu rozložení. Představíme-li si některou vlastnost síťového provozu (například intervaly mezi příchody paketů) jako řadu hodnot a začneme-li přes tuto řadu počítat klouzavý průměr s různě velkým oknem, pak u Poissonova rozložení se očekává, že čím většího okna se použije, tím více bude výsledná řada hodnot hladší a tím více se bude snižovat rozptyl dat. U síťového provozu se rozptyl pro zvětšující se okno nesnižuje. Tento poznatek vedl k závěru, že většina vlastností síťového provozu se řídí rozložením s tzv. těžkým koncem (*heavy-tail* [49]). *Heavy-tail* rozložení pravděpodobnosti se svým tvarem asymptoticky blíží hyperbole.

$$p(X > x) \sim x^{-\alpha}, x \to \infty, 0 < \alpha < 2.$$

Heavy-tail rozložení předpovídá, že převážná většina naměřených hodnot bude nízká, ale zaroveň existuje malá skupina vzorků s velmi vysokou hodnotou, která tvoří významnou část celkové sumy hodnot vzorků. Existují mnohé příklady *heavy-tail* rozložení. V ekonomii se využívá k popisu rozložení bohatství mezi populaci. Odtud pochází i známá poučka, že 80% bohatství je vlastněno 20% populace. Toto rozložení bylo identifikováno v mnoha dalších oblastech, například v rozsahu lesních požárů, velikosti částic v písku nebo velikosti souborů na pevném disku. V souvislosti se síťovým provozem bylo *heavy-tail* rozložení pozorováno u různých typů provozu, jako je webový provoz (velikost stránek), FTP (velikost souborů), TELNET (délka sezení), ale i na nižších úrovních modelu ISO/OSI jako je TCP (velikost toku dat).

Usuzuje se, že příčina *heavy-tail* rozložení vlastností síťového provozu je pravděpodobně způsobena rozložením velikosti souborů [45] , chováním uživatelů či aplikací [70, 21] a dynamikou síťového provozu [48, 40, 73, 32]. *Heavy-tail* rozložení charakteristik tak představuje zcela nový problém při analýze a zpracování síťového provozu, kdy nelze využít dříve používaných předpokladů a vznikají nebo zanikají možnosti optimalizací za účelem zrychlení nebo zefektivnění zpracování síťového provozu.

3.2 Složení provozu a sledované charakteristiky

Nejdůležitější sledované charakteristiky pro jeden tok síťového provozu jsou šířka zabraného přenosového pásma, počet paketů za daný časový úsek, velikosti paketů, ztrátovost paketů, doba oběhu (Round Trip Time). Při sledování celkového obrazu provozu se sledují nejen tyto charakteristiky, ale navíc se sledují statistiky o počtu nově vzniklých toků za daný časový úsek či rozložení toků mezi aplikace. Právě aplikace ovlivňují nejvíce složení síťového provozu.

V průběhu mnoha let vývoje Internetu nastaly tři význačné éry [56]. První z nich lze nazvat érou textových aplikací, kdy většina provozu náležela emailové komunikaci, přenosům souborů a USENET skupinám. V tomto období bylo 80% provozu přenášeno přes TCP (většinou SMTP a FTP) a zbylých 20% patřilo UDP (DNS, Telnet) [11, 12]. Následovala éra hyperlinková, kdy podíl HTTP provozu vzrostl z malého procenta až na 75% celkového provozu a stejně tak TCP dosáhlo svého současného podílu, tj. 80% až 90% provozu, zbytek náleží UDP [11]. Poslední éra přetrvávající dodnes se nazývá multimediální, neboť provozu dominují multimediální aplikace (hovory, videokonference, sdílení multimediálních souborů) používající P2P[1] paradigmatu. V nedávné době byl podíl P2P aplikací v provozu vyšší než u HTTP (asi 55% bajtů a u HTTP zhruba 40% bajtů). Nicméně v současné době je HTTP provoz odpovědný za velké množství přenášených dat i toků. 38% toků (60% dat) náleží HTTP oproti pouhým 4% toků (20% dat) u P2P [23, 4].

Přestože nejčastěji používané aplikace jsou známé, není možné přímo odvodit výsledné vlastnosti provozu. Tyto vlastnosti jsou ovlivněny dalšími faktory jako jsou topologie sítě, chování uživatelů a další. Vznikly ovšem modely složení síťového provozu podpořené měřením provozu na reálné síti. Mezi nejvýznamnější z pohledu této práce patří klasifikace toků dle jejich velikosti, doby trvání a intenzity.

3.3 Třídy toků a jejich vlastnosti

První z prací, která se věnovala vlastnostem toků, byla publikována již v roce 1986 [32]. Autoři zjistili, že pakety toku přicházejí ve shlucích, kdy po období dlouhých intervalů mezi pakety může následovat období velice krátkých intervalů. Tento proces může být pozorován ve více časových úrovních. To můžeme považovat za první důkaz soběpodobnosti síťového provozu. Další práce se zabývaly distribucí velikosti toků ve smyslu přenesených dat a paketů, intenzitou nebo trváním toku. Všechny konstatují závěr, že zkoumané vlastnosti se řídí *heavy-tail* rozložením. Význačné toky z obou konců rozložení dostaly i jednotlivá pojmenování. Podle množství dat v toku byly rozděleny na slony a myši, kde slony tvoří jen malé množství velkých toků, jenž přenáší většinu dat a naopak myši tvoří velké množství malých toků, které přenáší výrazně méně provozu. Kromě množství dat byly toky pojmenovány i podle dalších vlasností. Z pohledu délky trvání byly rozděleny na krátké (vážky) a

[1] peer to peer

dlouho-žijící (želvy). Z pohledu intenzity na rychlé (gepardy) a pomalé (šneky). A konečně podle shlukovosti na toky s velkou shlukovostí (dikobrazy) a malou (rejnoky).

V několika publikacích [57, 26, 9] byli studováni sloni a myši. Výsledky potvrdily, že převážná většina provozu ve smyslu bajtů nebo paketů je přenesena malým počtem toků, zatímco existuje mnoho toků, které tvoří jen malé procento celkového provozu. Například v [56] autoři využili několikaletý vzorek provozu mezi Japonskem a západním pobřežím Spojených států, aby demonstrovali tuto skutečnost.

Existence velkého množství malých toků byla jasně prokázána například v článku [18], který analyzuje síťový vzorek osahující až 40% toků s jediným paketem.

Následně článek [10] věnovaný analýze délce trvání toků reportuje, že 45% toků trvá méně než 2 sekundy, 53% trvá mezi 2 sekundami a 15 minutami a 12% trvá déle. Studie síťového provozu aktuálnějšího vzorku dat ze sítě MAWI ukázala, že až 70% toků je kratších než 10 s. Měření propustnosti toku se věnoval Heidemann [39] a zjistil, že 80% toků jsou šneci s propustností menší než 10 kB/s, a jen 2% toků gepardi s více jak 100 kB/s.

Savrotham [55] ukázal, že shlukové chování je způsobeno jen několika toky s velkým objemem dat a je daleko pravděpodobnější, že tyto toky jsou odpovědné za blokování linky. Jeho výsledky jsou konzistentní i s výsledky v [39], přestože oba autoři definovali shlukovost odlišně.

Jednotlivé rozdělení toků do tříd vzniklo v různých publikacích podle toho, kterou vlastnost daná publikace zkoumala. Ucelený obraz o vlastnostech toků a jejich vzájemné korelaci podal až Heideman [39]. Zároveň podává i vysvětlení, co danou vlastnost a její korelaci k ostatním vlastnostem způsobuje a jak aplikace a uživatelé ovlivňují chování síťového provozu.

Tabulka 3.1: Vzájemný vztah mezi slony, želvami, gepardy a dikobrazi (převzato z [39]).

	Sloni	**Želvy**	**Gepardi**	**Dikobrazi**
Sloni	-	6%	3%	68%
Želvy	20%	-	0.007%	8%
Gepardi	7%	0.004%	-	3%
Dikobrazi	19%	1%	4%	-

Tab. 3.1 ukazuje vzájemný vztah mezi různými třídami toků na dvou vzorcích síťového provozu. Některé toky nelze klasifikovat ve všech třídách, proto součet procent v tabulce neodpovídá 100%. Mezi významné výsledky patří zjištění, že převážné procento dikobrazů (70%) jsou sloni a že existuje silná korelace mezi velkými a shlukovými toky. Výsledky změřené na obou vzorcích se výrazně liší pro korelaci dikobrazů a gepardů, což autoři vysvětlují slabým DNS provozem v prvním vzorku dat a vyvozují, že druhý vzorek dat obsahuje shlukové DNS toky, které tvoří 60% gepardů.

Tab. 3.2 zobrazuje pět nejvýznamnějších aplikací pro každou třídu. Konkrétní aplikace je určena pouze na základě čísla cílového portu přiřazeného organizací IANA. To může vést ke zkreslení výsledků zejména pro P2P aplikace, jež používají čísla portů jiných aplikací (snaží se tak maskovat svůj provoz) nebo dynamicky přidělované. Je vidět, že HTTP a P2P provoz tvoří většinu provozu, což je potvrzení výsledků prezentovaných v [54]. Heidemann uvádí, že webový provoz je odpovědný za většinu rychlého a shlukového přenosu. Více než 50% dlouho trvajících toků náleží DNS provozu, jehož toky se navíc chovají jako gepardi a dikobrazi zároveň. Tab. 3.3 přehledně shrnuje vlastnosti význačných toků a jejich chování,

Tabulka 3.2: Zastoupení prvních čtyř aplikací ve význačných tocích (převzato z [39]).

Sloni	**Želvy**	**Gepardi**	**Dikobrazi**
web (67%)	DNS (51%)	web (53%)	web (71%)
kazaa (5%	web (15%	DNS (28%)	SMTP (10%)
telnet (3,5%)	telnet (9,1%)	ftp (5%)	ftp (6%)
gnutella (2%)	ftp (5%)	smtp (3,3%)	nntp (2,1%)

které bylo odvozeno z mnoha pozorování.

Tabulka 3.3: Chování význačných toků (A značí ano, odpovídají charakteristice, N značí ne, neodpovídají charakteristice, tabulka byla převzata z [39]).

Kategorie	**velikost**	**doba**	**propustnost**	**shlukovost**
Sloni	A	A	N	N
Želvy	N	A	N	N
Gepardi	N	N	A	A
Dikobrazi	A	N	A	A

Více než 95% gepardů jsou krátké toky trvající méně než jednu sekundu. Asi 50% sloních toků trvá přes dvě minuty, což naznačuje, že sloni žijí dlouho, nicméně pouze 6% sloních toků jsou želvy. Nízká korelace želv a slonů je pravděpodobně způsobena rozhodnutím uživatelů nestahovat velké objemy dat přes linky s nízkou propusností. Velké procento dikobrazů (65%) žije méně než deset sekund a 95% žije méně než dvě minuty. Vzhledem k tomu, že mezi dikobrazi a slony existuje pozorovatelná korelace, je převážná část dikobrazů považována za sloní toky, které jsou kratší než 2 minuty. Na základě tohoto pozorování Heideman usuzuje, že většina dikobrazích toků je způsobena přenosy velkých souborů po rychlých linkách, což je konzistentní výsledek s [55]. U gepardů je možné pozorovat, že jsou krátké a malé a dobře korelují s dikobrazi, kde tvoří doplněk ke sloním tokům, které nejsou tak rychlé.

Podle různých studií se nároky na propustnost linek zvyšují o 40–60% ročně [46, 2]. Počet uživatelů Internetu roste průměrně každý rok o 10% [3], hlavní podíl tak na růstu objemu přenesených dat mají nově vznikající aplikace (například IPTV, mobilní aplikace a další). Předpovědi růstu internetového provozu neočekávají, že by se rostoucí trend měl v příštích letech zastavit. Lze navíc očekávat vznik nových protokolů podmíněný zastaralostí některých protokolů pro moderní sítě a aplikace.

3.4 Použité datové sady a jejich charakteristiky

Dostupnost sad obsahující reálný síťovy provoz je velmi špatná, neboť se potýká nejen s technickými problémy, ale i s ochranou osobních dat uživatelů. Prvním problémem je sběr samotných dat na síti, kde stávající infrastruktura nedovoluje připojit specializovaná zařízení pro sběr dat. Na některých zařízeních je možné duplikovat provoz na tzv. SPAN port směrovače nebo přepínače. Data poskytovaná na SPAN portu jsou často nekvalitní, neobsahují veškerý původní provoz a sbírané pakety jsou často poškozeny. Důvodem je vysoké zatížení samotného prvku se SPAN portem, jež plní primárně jiné úkoly. Navíc kopírování

Tabulka 3.4: Popis a velikosti použitých datových sad.

Název	Délka [s]	Počet bajtů	Počet paketů	Počet toků
Mawi-2010/04/14-14:00	900 s	32 034 254 427	44 599 795	1 052 546
Snjc-2009/07/17-13:00	300 s	83 710 238 274	137 513 466	8 347 616
Vut-2011/10/18-15:00	300 s	47 711 413 780	58 541 612	2 545 287

provozu z několika linek na SPAN port způsobí zahlcení linky samotné. Další možností je vložit sběrný prvek přímo do linky. Vložení prvku, který není přímo spojený s provozem sítě, je považováno v produkčním prostředí za riziko. Do linky se tak vkládají rozbočovače, které z podstaty své konstrukce nemohou provoz ovlivnit. K těmto rozbočovačům je připojeno samotné sběrné zařízení. Dalším úskalím je samotné zařízení pro sběr paketů. Bez speciální podpory není zařízení schopné spolehlivě zachytit veškerý provoz na síti a nasbíraná datová sada nemusí odpovídat provozu na lince. Při poskytování nasbíraných dat vyvstává otázka s ochranou soukromí a osobních údajů uživatelů, jejichž provoz je ve vzorku dat zachycen.

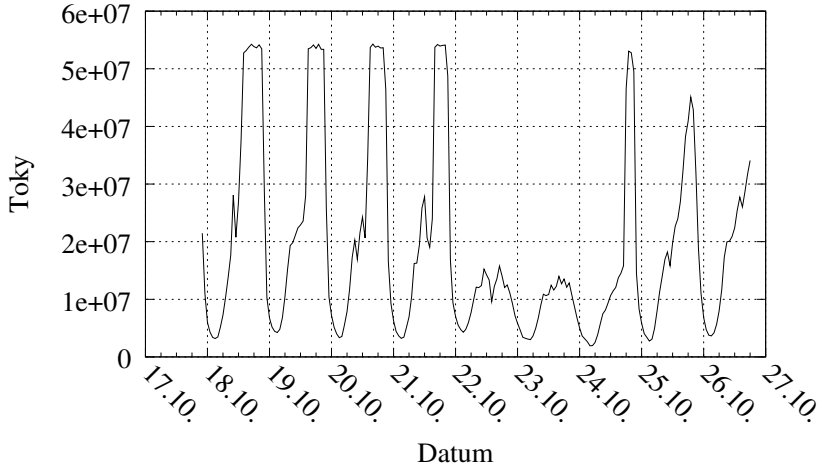
Tato práce využívá pro analýzu, návrh správ paměti a ověření výsledků celkem tři datové sady z různých sítí. Primární datovou sadou, která je používána a prezentována ve většině experimentů, je veřejně dostupná sada z archivu Mawi [4]. Jedná se o část vzorku provozu z trans-pacifické 155 Mb/s linky sítě Mawi samplepoint-F z období 13.-16.4.2010. Tento vzorek obsahuje oba směry komunikace v jednom souboru, proto zabraná kapacita linky přesahuje 150 Mb/s a pohybuje se kolem 285 Mb/s. Další vzorek dat se podařilo na vyžádání získat od americké organizace CAIDA [68], která se zabývá sběrem a analýzou dat na reálné síti. Jedná se o vzorek provozu z 10 Gb/s páteřní linky v San Jose ze dne 17.7.2009 s průměrným zatížením kolem 2,5 Gb/s. Jedná se pouze o vzorek provozu z jednoho směru linky. IP adresy v datových sadách *Mawi-2010/04/14-14:00* a *Snjc-2009/07/17-13:00* jsou anonymizovány. Tato anonymizace zachovává příslušnost paketu k toku a proto nijak neovlivňuje experimenty ani charakteristiky, které jsou v rámci této práce sledovány. Posledním získaným vzorkem dat je provoz z obou směrů 10 Gb/s linky připojující část sítě VUT v Brně do sítě CESNET ze dne 18.10.2011 se zatížením do 1 Gb/s. Tabulka 3.4 uvádí velikosti datových sad, které jsou v dalších částech této práce využívány pro experimenty.

Díky sledování toků pomocí NetFlow infrastruktury na síti VUT v Brně je možné rovněž pozorovat průběh různých vlastností síťového provozu v průběhu delšího období. Z pohledu této práce je nejzajímavější vlastností počet toků, které se v síťovém provozu vyskytují. Obr. 3.1 ukazuje graf počtu toků, které byly nasbírány během jedné hodiny v průběhu jednoho týdne. Je dobře viditelné, že průběh má silnou 24-hodinovou periodu. Nejvyšších hodnot dosahuje kolem 14. až 15. hodiny zatímco nejnižších po 3. hodině ráno. Průběh je silně ovlivněn složením uživatelů (v síti VUT jsou to převážně studenti). U zbylých dvou sad *Mawi-2010/04/14-14:00* a *Snjc-2009/07/17-13:00* jsou k dispozici statistiky pouze z několika dnů. Lze konstatovat, že v komerčních sítích jsou vrcholy grafu posunuty více do pracovního dne a útlum provozu přichází již před 10 hodinou. Z tohoto důvodu jsou pro experimenty používány datové sady nasbíráné v časovém úseku mezi 13. až 15. hodinou, kdy je provoz nejvíce intenzivní. Množství provozu klesá přes víkend (22.-23.10.), kdy velká část studentů opouští koleje. Ze stejného důvodu není žádná z datových sad z časového úseku během víkendu. Počet současně aktivních toků se liší dle použitých datových sad. Tabulka 3.5 dává představu o jejich množství.

Velmi zajímavým údajem je rozložení velikosti toků $|f_i|$ v datových sadách. Jak uvádí

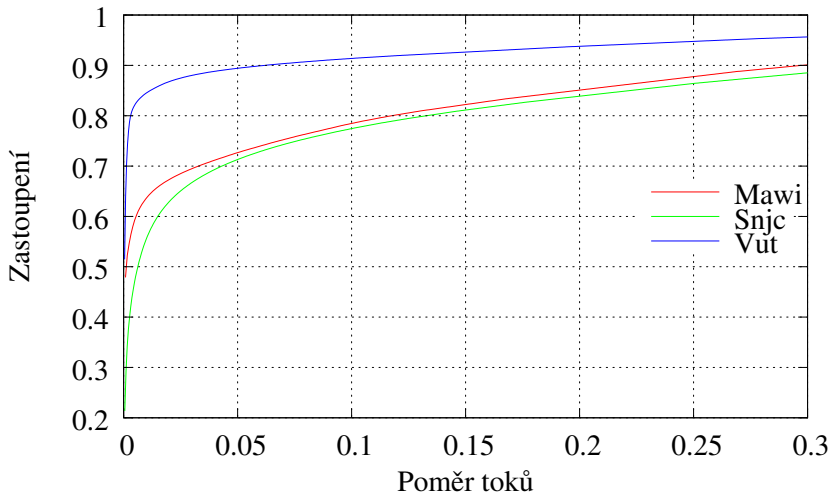
Tabulka 3.5: Statistiky současně aktivních toků v datových sadách.

Název	Průměr	Minimum	Maximum
Mawi-2010/04/14-14:00	67 315	56 592	250 165
Snjc-2009/07/17-13:00	1 784 392	1 716 438	1 803 447
Vut-2011/10/18-15:00	114 057	94 925	148 393



Obrázek 3.1: Průběh počtu toků za hodinu v průběhu týdne na sledovacím místě VUT v roce 2011.

citovaná literatura, většina toků by měla být velmi malých a jen několik by mělo být velmi velkých. Zároveň by ale měly velké toky přenášet většinu síťového provozu. Tento jev je pozorován i v datech použitých v této práci a je dobře patrný z obr. 3.2. Kumulativní distribuční funkce množství paketů dle velikosti toků na obr. 3.2 je sestrojena následovně. Nejprve jsou velikosti toků sezařeny sestupně dle velikosti. Následně jsou velikosti postupně ščítány a normalizovány celkovým počtem paketů v datové sadě. Průběžný součet normalizovaných hodnot odpovídá hodnotě na ose y, na ose x lze vyčíst poměr započítaných toků vůči celkovému počtu toků v datové sadě. U všech zobrazených datových sad je patrné *heavy-tail* rozložení velikostí toků. U datové sady *Vut-2011/10/18-15:00* odpovídá 1% největších toků za 84% přenesených paketů, v případě páteřních linek *Mawi-2010/04/14-14:00* a *Snjc-2009/07/17-13:00* obsahuje 1% největších toků kolem 60% přenesených paketů. Deset procent největších toků odpovídá za téměř 80% procent přenesených paketů a 95% přenesených bajtů. Složení provozu na páteřních linkách se liší od provozu získaného ze specifické sítě VUT. Vysoká propusnost připojení VUT do Internetu dovoluje přenášet velké objemy dat spojené s přenosem velkých souborů. Naproti tomu páteřní linky obsahují smíšený provoz od různých poskytovatelů Internetu koncovým zákazníkům (firmám nebo jednotlivcům). Připojení koncových zákazníku do Internetu je většinou omezeno cenou, kterou jsou ochotni zaplatit za poskytnutou propusnost. Ve shodě s předchozími studiemi je možné usuzovat, že uživatel si je vědom limitů připojení a přizpůsobí mu i styl práce s daty na síti, tj. přenáší menší velikosti dat. Obr. 3.3 zobrazuje normalizovaný kumulativní histogram vzdáleností po sobě následujících paketů stejného toku. Pokud se jedná o poslední (anebo jediný) paket v toku, je vzdálenost k dalšímu paketu nastavena na maximální možnou vzdálenost v sadě, tzn. celkový počet paketů v sadě. Z obr. 3.3 je patrné,



Obrázek 3.2: Kumulativní distribuční funkce seřazených velikostí toků v datové sadě *Mawi-2010/04/14-14:00*, *Snjc-2009/07/17-13:00*, *Vut-2011/10/18-15:00*.

že převládájí malé vzdálenosti paketů. V sadě *Mawi-2010/04/14-14:00* a *Vut-2011/10/18-15:00* je ve více než 80% případů vzdálenost dvou paketů stejného toku od sebe menší než 10000 paketů. U datové sady *Snjc-2009/07/17-13:00* je to pouze 57%.

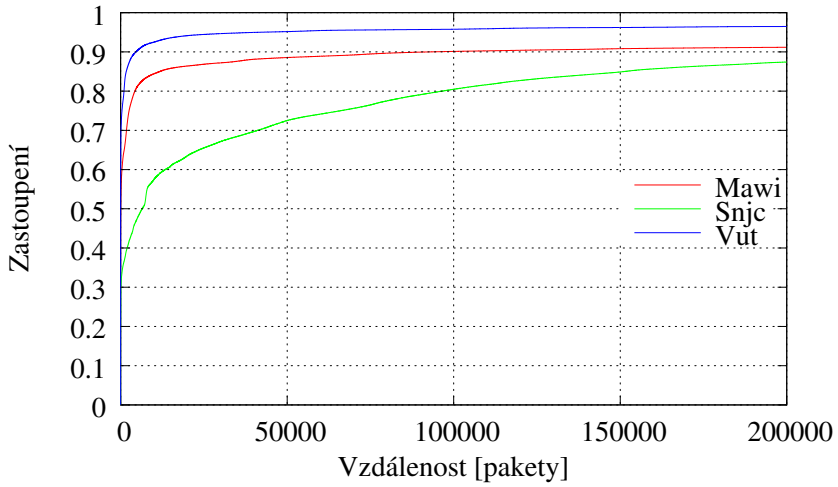
Zajímavým údajem je rovněž rozložení délek paketů. Histogram na obrázku 3.4 je vytvořen na základě datové sady *Snjc-2009/07/17-13:00* a ukazuje, že rozložení délek paketů v síťovém provozu není uniformní. V souladu s pozorováním jiných studií [57, 54] je rozložení délek paketů bimodální. Velká část paketů (60%) má délku do 200 B a další část (30% paketů) 1280 až 1500 B. Z pohledu rozložení délek paketů jsou všechny sady velmi podobné.

3.5 Úprava definice sloních toků

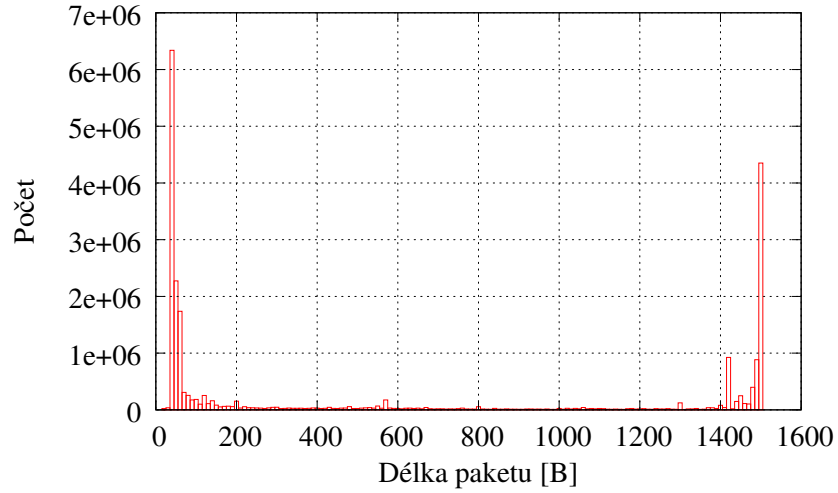
Pro účely této práce zavedeme úpravu definice sloních toků. Po této úpravě budeme sloní toky označovat jako velké a myší toky jako malé. Tok je shodně se sloním označen jako velký, pokud během své existence zabírá více než stanovené procento z přenosové kapacity linky. Pro vyloučení toků, které existují pouze krátkou dobu a na celkovém objemu dat nemají zásadní podíl, je zavedena penalizace v podobě minimální doby existence $t_{min} = 5$ s. To znamená, že u každého toku f s dobou existence $|f|_t$ kratší než t_{min}, je doba trvání prodloužena na pět sekund. Důvod zvolení pěti sekund je vysvětlen níže. U toků delších než pět sekund je ponechána původní doba trvání. Pro výpočet zabrané šířky pásma r_f tokem f s touto penalizací slouží následující vztah:

$$r_f = \frac{|f|_b}{\max(t_{min}, |f|_t)}. \tag{3.1}$$

Na základě hodnoty r_f jsou toky rozdělovány do kategorií dle velikosti. V této práci jsou využívány 4 kategorie: L_1, L_2, L_3, L_4. První L_1 obsahuje velmi velké toky s r_f vyšší než 0,1% kapacity linky, druhá L_2 obsahuje velké toky s $0{,}1\% \geq r_f > 0{,}01\%$ kapacity linky, třetí L_3 obsahuje středně velké toky s $0{,}01\% \geq r_f > 0{,}001\%$ a čtvrtá L_4 obsahuje zbývající malé toky s $r_f \leq 0{,}001\%$, které nejsou předmětem zájmu sledování. Volbou délky intervalu t_{min} je ovlivněn výběr toků do jednotlivých kategorií. Tato situace je znázorněna na obr. 3.5, 3.6.

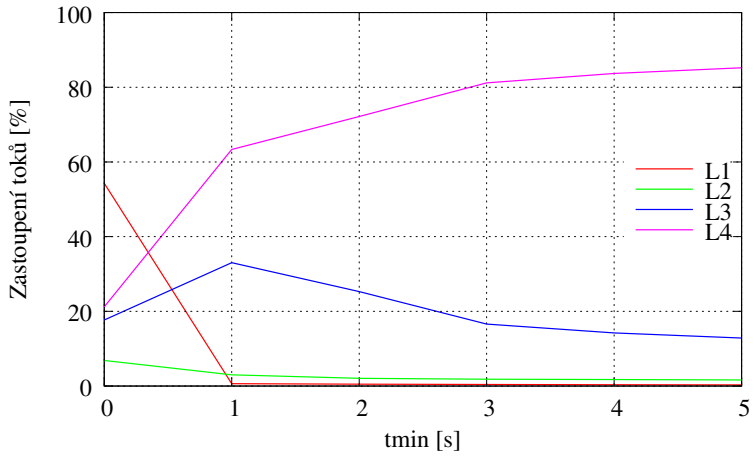


Obrázek 3.3: Kumulativní distribuční funkce rozložení vzdáleností paketů v rámci toků pro datové sady *Mawi-2010/04/14-14:00*, *Snjc-2009/07/17-13:00*, *Vut-2011/10/18-15:00*.

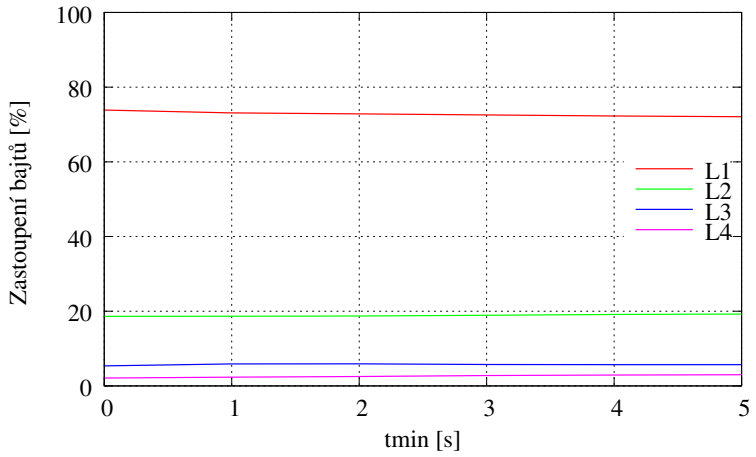


Obrázek 3.4: Histogram rozložení délek paketů v datové sadě *Snjc-2009/07/17-13:00*.

Obr. 3.5 popisuje závislost mezi množstvím toků vybraných do jednotlivých kategorií a délkou intervalu t_{min}. Obr. 3.6 popisuje závislost mezi množstvím přenesených bajtů jednotlivými kategoriemi toků a délkou intervalu t_{min}. Cílem je zvolit takový interval, který odstraní co možná nejvíce toků z prvních tří kategorií, ale zároveň zachová předpoklad, že velké toky přenáší většinu síťového provozu. Z obrázku 3.5 je patrné, že pokud neexisuje interval t_{min} ($t_{min} = 0$ s), pak může být až 55% toků považováno za velké. S prodlužujícím se t_{min} výrazně klesá procento velkých toků, nicméně z obrázku 3.6 je vidět, že dané toky nepřenáší velké množství dat. Jejich odstraněním z velkých kategorií pomocí penalizace se jen nepatrně snižuje objem přenesených dat v těchto kategoriích. Lze konstatovat, že volbou $t_{min} = 5$ s je výrazně sníženo procento velkých toků (součet prvních třech kategorií je asi 15%), které ale přenáší až 98% objemu dat. Přestože by bylo možné interval dále zvyšovat, úbytek toků by již nebyl tak významný. Navíc v návaznosti na předchozí práce [25, 24], které typicky pracují s toky v pěti-sekundových intervalech, se jeví vhodné zachovat toto pravidlo.

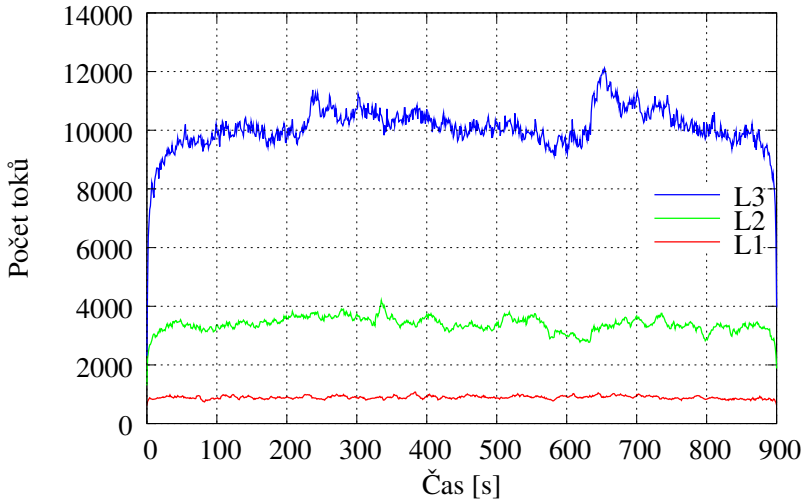


Obrázek 3.5: Množství toků patřící daným kategoriím pro zvyšující se t_{min}.



Obrázek 3.6: Množství bajtů patřící daným kategoriím pro zvyšující se t_{min}.

Na obrázku 3.7 je zobrazen typický průběh počtu současně aktivních toků v různých kategoriích v průběhu času pro datovou sadu *Mawi-14:15*. Na začátku počet toků rychle stoupá až dosáhne určité úrovně, kolem které osciluje. Vzestup toků na začátku intervalu je způsoben tím, jak se ve vzorku dat začínají postupně objevovat stále nové toky. Na konci intervalu počet toků postupně klesá. V této oblasti dochází k tomu, že i když by byl tok v budoucnu přiřazen například do L_2 nebo L_3, ke konci měřeného intervalu již neobdrží dostatek dat a je přiřazen do L_4. Analogicky na začátku intervalu postupně přibývají toky, které mohou být přiřazeny do kategorie L_2 a L_3. U největších toků L_1 tento trend není patrný, neboť velké toky typicky trvají velmi dlouho [39], a tím pádem existují již v počáteční fázi datové sady. Navíc jen několik velmi velkých toků začne v průběhu měřeného intervalu, proto se počet L_1 toků ke konci intervalu drží na stabilní hodnotě. Pro experimenty jsou opět použity datové sady *Mawi-2010/04/14-14:00*, *Snjc-2009/07/17-13:00* a *Vut-2011/10/18-15:00*. Rozdělení množství toků do jednotlivých kategorií je popsáno v tabulce 3.6. Tato tabulka zobrazuje procentuální zastoupení jednotlivých kategorií v provozu daného vzorku co do množství toků, paketů a bajtů.



Obrázek 3.7: Počet současně aktivních toků v prvních třech kategoriích.

Tabulka 3.6: Rozdělení toků do kategorií dle velikosti.

Mawi-2010/04/14-14:00	L_1	L_2	L_3	L_4	Celkem
Toky	0,23%	0,93%	9,43%	89,41%	10 mil.
Pakety	31,97%	18,92%	20,71%	28,4%	44 mil.
Bajty	68,35%	17,13%	9,22%	5,3%	32 mil.
Snjc-2009/07/17-13:00	L_1	L_2	L_3	L_4	Celkem
Toky	0,00%	0,02%	0,35%	99,63%	8 mil.
Pakety	0,36%	10,15%	17,07%	72,42%	137 mil.
Bajty	0,85%	24,01%	36,48%	38,66%	83 mil.
Vut-2011/10/18-15:00	L_1	L_2	L_3	L_4	Celkem
Toky	0,10%	0,49%	3,2%	96,21%	2,5 mil.
Pakety	55,64%	15,22%	13,38%	15,76%	58 mil.
Bajty	76,38%	14,74%	5,75%	3,13%	47 mil.

Kapitola 4

Správy paměti cache

Časová lokalita dat značí, že existuje velká pravděpodobnost znovupoužití dat, která byla použita v předchozím časovém období. Naopak data, která již nějakou dobu použita nebyla, nebudou použita ani v blízké budoucnosti. Časovou lokalitu lze vyjádřit pravděpodobnostním rozložením vzdáleností (počítáno jako množství přístupů k jiným položkám) mezi dvěma následnými přístupy k dané položce. Popularita dat vyjadřuje pravděpodobnost výskytu položek v datech. V mnoha případech platí, že pravděpodobnost výskytu položek není uniformní. Naopak existuje jen málo položek, které jsou populární, a obrovské množství položek, které jsou nepopulární.

Pokud data vykazují dobrou časovou lokalitu nebo jsou některé položky více populární než jiné, pak je možné využít hierarchii pamětí. Se správně zvolenou správou paměti je možné docílit vyřízení většiny požadavků přístupem do paměti na nejnižší úrovni (cache).

V následujících sekcích jsou krátce popsány správy, které byly doposud navrženy pro správu paměti nebo v cache v různých oblastech jako jsou správa cache procesoru (například [30]), správa překladu stránek nebo správa objektů ve *web* cache (například [53]).

4.1 Triviální správy paměti

First In First Out (FIFO) je jedním z nejjednodušších přístupů pro správu paměti cache. Položky v cache jsou uloženy ve frontě. Nové položky jsou vkládány na začátek fronty a při nedostatku místa v cache jsou položky odebírány z konce fronty. Jednoduchost implementace spočívá v tom, že samotné využití položek nemá žádný vliv na pozici položek ve frontě a není třeba přistupované položky jakkoliv přemísťovat. Položka je do fronty vložena při svém prvním použití a následně je odsouvána dalšími nově vkládanými položkami. Doba, po kterou vydrží položka v cache, je dána pouze počtem nově příchozích položek. Pokud od vložení položky přišlo tolik položek, kolik je kapacita cache, pak je položka z cache odsunuta. Lze prohlásit, že všechny položky dostanou možnost využít cache rovným dílem. Jediným ukazatelem pro odsun položek je čas prvního využití položky. Intuitivně bude tato strategie dobře pracovat s daty, která vykazují specifický typ využití položek v paměti. Například pokud po prvním přístupu k položce následují další přístupy po omezený interval, který je dán velikostí pamětí cache. Po skončení tohoto intervalu není položka dále využívána. Množství přístupů a doba aktivity všech položek by měla být uniformní. Takový typ využití položek je ovšem velmi ojedinělý, naopak většina datových sad na Internetu vykazuje odlišné chování, jak bylo ukázáno v kapitole 3. Implementace této strategie správy cache je velmi jednoduchá. Lze ji implementovat jednosměrně vázaným seznamem, kde nové položky jsou

vkládány na konec seznamu a při nedostatku místa v paměti jsou položky odebírány ze začátku seznamu.

Random, neboli náhodná správa paměti, uvolňuje náhodně vybrané položky při zaplnění cache. Lze ji vnímat jako aproximaci správy paměti FIFO. Čím déle je položka v paměti, tím větší je pravděpodobnost, že bude uvolněna. Výhodou je, že tato správa paměti nepředstavuje téměř žadnou zátěž z pohledu výpočetní nebo paměťové náročnosti až na kvalitní generátor pseudo-náhodných čísel. Generátor musí mít svou periodu shodnou s velikostí spravované cache, aby pravděpobobnost uvolnění všech položek byla uniformní a aby nedocházelo k opomenutí některých položek v případech, kdy generátor není schopen generovat některé hodnoty z rozsahu paměti.

4.2 Správa paměti využívající nedávnou historii přístupů

Least Recently Used (LRU) je široce rozšířená strategie správy paměti cache využívána v různých oblastech, například pro správu překladu stránek paměti, databáze nebo diskové vyrovnávací paměti. Oblíbenost LRU plyne z efektivity při správě paměti na běžných datových sadách a z jednoduché implementace. LRU pracuje následujícím způsobem. Když je paměť plná, tato strategie odsouvá z paměti položku, která byla nevyužita nejdelší dobu. Implementace LRU může být realizována obousměrně vázaným seznamem položek. Nové položky a rovněž i používané (přistupované) položky jsou přesouvány na začátek tohoto seznamu, zatímco nepoužívané položky zůstávají na konci tohoto seznamu a poslední z nich je odstraněna z paměti. Tato strategie výměny položek v paměti cache předpokládá, že paměť bude pracovat s daty, která mají velmi dobrou časovou lokalitu, neboť nové a nedávno použité odsouvají již dříve použité položky. V případě špatné lokality, například výskytu postupného náčítání nových položek, bude LRU neúčinná. Z tohoto důvodu vznikla vylepšení LRU.

Segmented LRU rozděluje seznam položek spravovaných pomocí LRU na dvě části, tzv. chráněnou a nechráněnou část. Bod rozdělení seznamu je určen místem, kam jsou vkládány nové položky. Chráněná část seznamu vede od začátku seznamu až po místo vkládání nových položek. Od tohoto místa do konce seznamu se nachází nechráněná část. Určení vhodného místa vkládání nových položek, tedy velikosti chráněné části, je závislé na použité datové sadě a velikosti cache. Správné nastavení je předmětem experimentů a konkrétního nasazení. Přesun místa vložení ze začátku do vnitřní části seznamu umožňuje z cache rychleji odstraňovat položky, které nemají žádnou lokalitu a jejichž přítomnost v cache je nežádoucí, neboť zabírají místo pro položky s dobrou lokalitou. Předpokládá se, že položky s dobrou lokalitou se dostanou do chráněné oblasti, kdežto ty se špatnou lokalitou zůstanou v nechráněné oblasti. O místo v chráněné oblasti tak budou soupeřit pouze položky s dobrou lokalitou.

Least Recently Used Twice (LRU-2) je strategie, která využívá pro rozhodnutí o uvolnění položky z cache předposlední přístup k položce, na rozdíl od LRU, která uvažuje pouze poslední přístup k položce. Obecně lze rozšířit LRU-2 až na LRU-K [47] techniku, tedy uchovávání a rozhodování se podle posledních K přístupů k položce. Strategie LRU-K pracuje následovně. Při příchodu nové položky a plné cache se nahradí položka s nejdelším intervalem mezi K-tým přístupem do cache a aktuálním časem. Položky, které nebyly alespoň K-krát využity, mají interval nastaveny na nekonečno. Pokud tedy existuje více položek s menším počtem než K přístupů, pak se rozhoduje o uvolnění mezi těmito položkami. Sníží se K o jedna a postupuje se podle předchozího scénáře až na $K = 1$, kdy je aplikována již známá strategie LRU neboli LRU-1. K vyhledání položky s nejdelším intervalem navrhují

autoři LRU-K využít vyhledávací strom. Na základě experimentů nad datovými sadami ukazují autoři LRU-K [47], že LRU-2 dosahuje dostatečně dobrých výsledků a implementace LRU-K, kde $K > 2$, přináší pouze malé zlepšení v porovnaní s nárůstem režie spojené s udržováním dlouhodobé historie.

History LRU (HLRU) [58] byla navržena nezávisle na LRU-K, ale je naprosto shodná s LRU-K. HLRU byla navržena jako strategie pro uvolňování položek z cache uchovávající webové stránky a objekty.

Two Queues (2Q) [35] implementuje filozofii správy LRU-2 ale s konstantní časovou složitostí. Naproti tomu LRU-2 má logaritmickou časovou složitost vyhledání položky pro odstranění, neboť položky musí být indexovány pomocí vyhledávácího stromu. 2Q využívá v nejjednodušší verzi dvě fronty A_1 a A_m. Fronta A_m je spravována jako LRU seznam a uchovává často využívané položky. Fronta A_1 je spravována jako FIFO a slouží pro identifikaci položek, které mohou být potenciálně často využity. Potenciálně často využitelné položky jsou přesunuty do A_m. Rozdělením spravovaného seznamu položek na dvě fronty připomíná tato technika SLRU. Rozdílem oproti SLRU je, že A_1 je spravována pomocí FIFO strategie. Pokud je překročena velikost A_m, pak nejsou přebývající položky přesunuty do A_1, nýbrž jsou rovnou uvolněny.

Exponential Smoothing (EXP1) [53] uvažuje podobně jako LRU-K posledních K přístupů k položce. Na rozdíl od LRU-K přikládá vyšší váhu nedávným přístupům. Konkrétně váha jednotlivých přístupů klesá exponenciálně s jejich vzdáleností od aktuálního přístupů k položce. Nejvíce vzdálený přístup má nejnižší váhu a aktuální přístup nejvyšší. Podstatou strategie je odhad frekvence budoucího využití položky na základě historie přístupů, tedy $W(t_i) = \frac{1}{\mu_i}$, kde W je odhad frekvence, t_i je interval od i-tého přístupu k položce do současnosti a μ_i je odhad průměru intervalů mezi přístupy. Tento μ_i je počítán klouzavým průměrem s exponeciálním oknem jako $\mu_i = \alpha t_i + (1-\alpha)\mu_{i-1}$ a $\mu_0 = \frac{1}{t_0}$, kde α je konstanta ovlivňující délku využívané historie. Autoři navrhují nastavit $\alpha = 0,1$, kdy dochází ke kombinaci nedávnosti a frekvence přístupů. Konkrétní způsob implementace této strategie autoři neuvádí. Lze předpokládat, že se až na výpočet frekvence neliší od LRU-2, neboť položky je nutné řadit a vyhledávat ve vyhledávacím stromě na základě aktuální hodnoty μ_i.

Low Inter-Reference Recency Set (LIRS) [33] je založena na sledování intervalu mezi přístupy k položce. Tento interval je nazván IRR (Inter-Reference Recency). IRR pro každou položku v cache vyjadřuje počet přístupů, které se vyskytly mezi dvěma po sobě následujícími přístupy k sledované položce. Těmito dvěma následujícími přístupy se má na mysli poslední a předposlední přístup k položce. Správa LIRS pak předpokládá, že položky s velkým IRR budou mít s velkou pravděpodobností i další IRR velké (další IRR je měřeno při dalším přístupu k položce). Na základě tohoto předpokladu LIRS odstraňuje položky s velkým IRR, pokud je potřeba místo v cache. Na rozdíl od LRU odstraní i takovou položku z cache, u níž doba od posledního přístupu je velmi malá, ale IRR je velké. LIRS tak nespoléhá na nedávnost přístupu, ale spíše na historii přístupů k položce. Autoři tím chtějí zamezit potížím LRU rozpoznat rychlé sekvenční procházení položek, které se již neopakuje a zamezit uchovávání takových položek v cache. Správa LIRS dělí položky na základě aktuální situace do dvou skupin LIR (nízké IRR) a HIR (vysoké IRR). Položky aktuálně označené jako LIR jsou udržovány v cache a nejsou odsunuty. Jsou to právě HIR položky, které jsou odstraněny z cache, pokud je třeba uvolnit místo novým položkám. Pokud vzdálenost posledního přístupu HIR položce je menší než k LIR, pak je jejich příslušnost ke skupině zaměněna. Autoři LIRS navrhli přístup, jak lze tuto správu efektivně implementovat pomocí dvou seznamů spravovaných jako LRU. V jednom seznamu jsou uchovávány

LIR položky a ve druhém seznamu jsou uchovávány HIR položky. Implementací tato správa připomíná výše popsanou 2Q správu až na odlišnou správu seznamu s HIR položkami. Ta se jeví jako klíčová, neboť dle prezentovaných výsledků dosahuje LIRS lepších výsledků napříč všemi vygenerovanými testovacími daty.

4.3 Správa cache využívající četnost přístupů

V prostředí, kde množina často přistupovaných položek zůstává stále stejná nebo se mění jen velmi pomalu, je možné spravovat položky dle jejich popularity (počtu přístupů). Předpokládá se, že aktuálně populární položky budou populární i v budoucnu. Dalším předpokladem je, že celkový součet přístupů k populárním položkám bude vyšší než u nepopulárních. Úkolem správy cache je v takových případech sledovat počet přístupů k položkám a na základě této informace rozhodovat o tom, které položky z cache odstranit a které uchovat.

Least Frequently Used (LFU) se řadí mezi základní a zároveň nejznámější správy, které jsou založené na sledování četnosti přístupů k položkám. Pokud je cache plná a je potřeba zajistit místo pro novou položku, pak LFU odstraní položku, která byla nejméně často používána (položku s nejmenším počtem přístupů). Implementace této správy vyžaduje počítat přístupy ke všem položkám a řadit položky dle těchto hodnot. Při nedostatku volné kapacity je třeba vyhledat položku s nejnižší hodnotou čítače přístupů a tuto položku odstranit. Nevýhodou tohoto přístupu je, že cache může být „znečištěna“ položkami, které byly velmi populární v minulosti, ale nyní již populární nejsou.

*LFU** [6] je modifikací LFU. LFU* povoluje, na rozdíl od LFU, odstranit z cache pouze položky s jedním přístupem. Pokud v cache není místo a nejsou dostupné položky s jedním přístupem, pak se nová položka do cache nevloží. Arlitt a Wiliamson ve své práci [6] uvádějí, že tato správa cache má na vzorku dat z webového serveru horší výsledky než samotná LFU, nicméně je základem pro další modifikaci LFU, konkrétně LFU*-Aging, která již dosahuje výrazně lepších výsledků než samotná LFU či LFU*. Problém LFU nastává při stavu, kdy se cache zaplní položkami s četností vyšší než jedna a není možné do cache další položky přidávat. U většiny datových sad dochází k obměně populárních položek s časem a tím pádem položky v cache nebudou aktivní a přesto je nebude možné odstranit.

LFU-Aging neboli LFU se stárnutím položek. LFU-Aging se snaží odstranit neschopnost LFU rozpoznat situaci, kdy položka získala vysokou míru popularity v minulosti, nicméně v současné době již není používána. Tyto položky pak zůstávají v cache zbytečně dlouho a zabírají tak místo novým položkám. LFU-Aging odstraňuje tento jev zavedením tzv. stárnutí a mezní hodnotou čítače přístupů, které může položka dosáhnout. Ke stárnutí dochází následovně. Pokud hodnota celkového čítače přístupů přesáhne stanovený limit, pak je hodnota čítače snížena na polovinu. Limitní hodnota čítače by měla být nastavena na základě analýzy vlastností prostředí, ve kterém je cache použita.

LFU-Aging* [59] kombinuje LFU* a LFU-Aging. LFU*-aging zabraňuje přílišné prioritizaci frekventovaně přistupovaných stavů omezením maximální hodnoty čítače přístupů. Dále odstraňuje položky s maximálně jedním přístupem. Pokud taková položka není k dispozici, pak nevloží vůbec novou položku do cache. A třetí vlastností je stárnutí uložených stavů pomocí zmenšení hodnoty čítačů na polovinu po uplynutí určitého intervalu. Dle Arlitta dosahuje tato kombinace vlastností výrazně lepších výsledků oproti běžnému LFU, i když samostné LFU* a LFU-Aging dosahují horších výsledků.

Window-LFU [31] je modifikací LFU, kdy měření počtu přístupů k položkám v cache je limitováno na určitý počet přístupů do cache. To znamená, že počet přístupů k položce je

počítán pouze z posledních několika přístupů, na rozdíl od LFU, kdy jsou počítány přístupy od vzniku položky. Nevýhodou je paměťová náročnost udržování průběžné historie přístupů v cache.

LFU-DA [22] neboli LFU with Dynamic Aging (LFU s dynamickým stárnutím). LFU-DA se snaží odstranit problém s nastavením parametrů u LFU-Aging. Nastavení parametrů je závislé na datové sadě, se kterou cache pracuje, a je nutná analýza této datové sady pro správné nastavení parametru stárnutí a mezní hodnoty čítače. V některých případech se vlastnosti datové sady mohou měnit a není možné dopředu předpočítat zmíněné parametry. LFU-DA proto zavádí tzv. *inflation factor* (faktor běžícího stáří cache) L. Faktor L má na začátku hodnotu nula, a v průběhu uvolňování položek je aktualizován na $L = K_i$, kde K_i je prioritní klíč uvolněné položky i, který se spočítá jako:

$$K_i = F_i + L_i,$$

kde F_i je počet přístupů k položce i. Hodnota L_i je uložena do každé položky v okamžiku jejího vložení do cache a je rovna hodnotě L v daném okamžiku. Rozhodnutí, které položky uvolnit, je prováděno na základě prioritního klíče K_i a nikoliv pouze na základě frekvence F_i. Tím, jak se L zvyšuje při odstraňování položek z cache, získávají nedávno uložené položky vyšší hodnotu K_i, což vede k tomu, že časem překonají staré populární položky, které jsou nakonec z cache odstraněny.

Value-aging [72] odhaduje frekvenci přístupů k položce od jejího vzniku akumulací intervalů mezi přístupy, kdy poslední interval má největší váhu. O uvolnění dané položky se rozhoduje na základě ohodnocení V, které je při přístupu k položce aktualizováno dle následujícího výrazu:

$$V_{new} = V_{old} + C_t\sqrt{\frac{C_t + L_t}{2}},$$

kde C_t je aktuální čas a L_t je čas posledního přístupu k položce. Položka s nejnižší hodnotou V je uvolněna z cache.

Alpha-aging [72] zavádí funkci $f(V)$, která má za úkol periodicky snižovat hodnotu V u všech položek. V tomto případě se nová hodnota V_{new} vypočítá při přístupu k položce jako:

$$V_{new} = V_{old} + C_t,$$

kde C_t je opět aktuální čas. Funkce $f(V)$ je definována jako:

$$f(V) = \alpha V, 0 \leq \alpha < 1$$

a je aplikována na hodnotu V periodicky s definovaným intervalem. Výraz uveden v literatuře vymezuje koeficient α na $\alpha > 1$.V takovém případě budou položky zvyšovat svou hodnotu V, i když nejsou používány a zůstávají tak v cache. Koeficient α by tedy měl být omezen na $0 \leq \alpha < 1$.

4.4 Adaptivní správa cache

Nedávno bylo navrženo několik algoritmů správy cache, které sledují chování bloků několika položek, tedy přístup k těmto položkám, co se týče časové a prostorové lokality. Na základě tohoto chování se snaží určit, který algoritmus použijí pro správu daného bloku, aby postihly specifické nároky různých programů a dosáhly lepšího využití cache.

Správa *SEQ* [27] navržená autory Glass a Cao realizuje adaptivní výměnu stránek pro správu virtuální paměti. Navržený algoritmus se nazývá SEQ a detekuje vzory v sekvencích adresových referencí. Pokud jsou nalezeny dlouhé sekvence adres způsobující výpadky stránek, pak je aplikována správa *Most recently used* (MRU). MRU odstraňuje z cache položky, které byly nedávno použity. Předpokladem dobré práce MRU je neexistence časové lokality u těchto položek. Ve všech ostatních případech je aplikována správa LRU. Autoři předpokládají, že LRU je dostatečně dobrá správa, která selhává pouze v případech, kdy přístupy ke stránkám v paměti vykazují specifické vzory přístupů, například sekvence přístupů k položkám se zvyšující se adresou. SEQ je založen právě na detekci takových sekvencí. To snižuje jeho univerzálnost, neboť existují i další vzory chování, u nichž může LRU selhat.

Adaptive Replacement Cache (ARC) [43] je adaptivní správa, která se snáží zobecnit detekci specifických sekvencí přístupů k položkám v cache. Změna správy cache je řízena na základě detekce mnoha odsunutých stránek, které byly nedávno vloženy do cache. Implementace této myšlenky spočívá ve sběru informace o distribuci výpadků stránek a následném propočtu, v jakém bodu se vyplatí využít jinou správu paměti oproti LRU. Část položek je pak spravována LRU a část pomocí MRU. Nevýhodou ARC je pomalá reakce na změnu přístupů ke stránkám. To je způsobeno tím, že velikost části paměti, která je spravována alternativní správou, je určena na základě celkové distribuce přístupů k položkám, což způsobí skrytí rychlých změn.

Detection based Adaptive Replacement (DEAR) [16] je rozšířením správy paměti SEQ. DEAR algoritmus je založený na klasifikaci řetězců přístupů do paměti. Pro každou běžící aplikaci DEAR určuje, zda se jedná o sekvenční, kruhový, shlukový nebo náhodný přístup. Na základě klasifikace DEAR použije pro každou aplikaci příslušnou správu paměti. Pro sekvenční a kruhový vzor chování aplikuje DEAR správu MRU, pro shlukový vzor LRU a pro náhodný vzor LFU.

Unified Buffer Management (UBM) [37]. UBM je pokračovatelem DEAR. Oproti DEAR navíc automaticky určuje velikost částí cache přiřazené jednotlivým vzorům chování. Toto přidělování je řízeno očekávaným ziskem při alokaci dalšího místa dané správě paměti.

4.5 Genetické algoritmy a cache

Princip genetického algoritmu je dobře popsán v knize D. Golberga [28]. V této práci jsou zmíněny jen základní prvky tohoto algoritmu. Základní myšlenkou GA je vývoj populace jedinců, kteří svým chromozomem reprezentují kandidátní řešení problému. Vývoj probíhá cyklicky, kdy v průběhu každého cyklu jsou jedinci v populaci ohodnoceni z hlediska vhodnosti pro řešení daného problému. Následně jsou vybrána slibná řešení, která jsou genetickými operátory křížení a mutace zkombinována a pozměněna. Nově vzniklí jedinci jsou ohodnoceni a začleněni do nově vznikající populace. Tím je cyklus uzavřen. Vývoj skončí při dosažení ukončující podmínky (například počet generací, malé zlepšení mezi generacemi).

Genetické algoritmy byly využity pro řešení mnoha optimalizačních problémů. Mezi jinými byly GA aplikovány i na optimalizaci paměti cache v oblasti procesorů. V práci [5] jsou pomocí genetického algoritmu hledány indexy bloků dat, které mají být z paměti uvolněny, v závislosti na vzoru přístupů do paměti. Jedná se tedy o způsob, jak nastavit správu paměti ARC [43] podle historie nedávných přístupů. Například uvažme, že vzor přístupů byl: „úspěch, úspěch, neúspěch", zakódovaný jako $110_2 = 6_{10}$. Pro uvolnění z paměti je vybrán z LRU seznamu blok dat, na který ukazuje šestá položka vyvinutého

chromozomu. Pokud je zvolena první položka v LRU seznamu, pak se jedná o správu MRU. Pokud je zvolena poslední položka, pak se jedná o LRU.

V některých případech může být výhodné zcela potlačit správu cache pro účely pevné délky běhu programu (například v systémech pracujích v reálném čase). Cílem je nalézt takové části kódu či pevných dat, které je možné nahrát a uzamčít v cache a zároveň dosáhnout co nejkratší ale pevné doby běhu programu. K nalezení těchto částí je navržen v [13] genetický algoritmus.

Mezi další aplikace evolučních algoritmů v oblasti cache patří nalezení specifického mapování položek do cache takovým způsobem, aby se u semi-asociativních pamětí dosáhlo co nejlepší zaplněnosti cache. Kaufmann a spol. [36] využili kartézské genetické programovaní pro vývoj mapovací funkce z množiny adres bloků do adres v cache.

4.6 Optimální správa cache

Reálné správy (prezentované v sekcích 4.2, 4.3 a 4.4) pracují bez znalosti budoucích přístupů k položkám v paměti. Tyto správy typicky pracují nad proudem dat a ze své podstaty jsou vždy založeny na heuristice. Heuristika může pracovat pouze s daty, která již daným zařízením prošla v minulosti. Na základě této znalosti se heuristika snaží přiblížit výsledkům optimálního algoritmu, který má znalost o přístupech a velikostech budoucích položek. Obecně je nalezení řešení optimálního algoritmu NP-úplný problém.

V případě shodné velikosti položek se problém výrazně zjednoduší. Při nedostatku místa v cache postačí vybrat takovou položku, ke které nebude v budoucnu nejdéle přistoupeno. *Farthest In The Future (FITF)* je algoritmus navržený v práci [8]. Tento algoritmus si během svého prvního průchodu daty zaznamená časy přístupů ke všem existujícím položkám. V druhém průchodu značí u položek v cache, kdy bude položka v budoucnu znovu přistoupena. U položek, které v budoucnu neobdrží žádný přístup, je čas přístího přístupu nastaven na nekonečno. V případě plné cache je vybrána položka, která má nejvzdálenější čas svého příštího přístupu.

4.7 Filtry

Vzhledem k obrovskému množství síťových toků, které se současně vyskytují na přenosových linkách, je velmi náročné sledovat všechny toky. Autoři Estan a Varghese [25] proto navrhli aplikovat filtr před samotným sledováním síťových toků. Toto rozšíření má za úkol odfiltrovat malé (z hlediska objemu přenesených dat) toky a propustit pro sledování pouze velké toky. Velkých toků je velmi málo, ale jsou odpovědné za většinu provozu. Pro jejich sledování by tak mělo stačit menší množství paměti, u které rychlost nepředstavuje problém. Na druhou stranu je problémem takové toky rozpoznat a uchovat v paměti dostatečně dlouho. Důvodem je obrovské množství zbývajících toků, které je nutné identifikovat a odstranit.

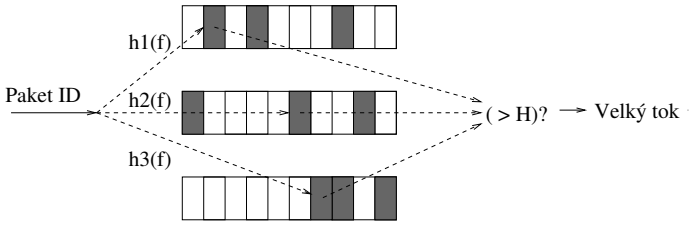
Nejjednodušším způsobem identifikace velkých toků pomocí filtrovacího přístupu je využití vzorkování na úrovni bajtů. Například bude-li nastavené vzorkování vstupního provozu na poměr 1:10000 (vzorkuje se paket obsahující 10000. bajt), pak jednopaketové toky mají jen malou pravděpobodnost, že jejich paket bude navzorkován. Naproti tomu tokům se vzrůstající velikostí roste šance na navzorkování alespoň jednoho z paketů.

Na vzorkování je založena i metoda Sample-and-Hold publikovaná v [25]. Oproti klasickému vzorkování přidává následující modifikaci přidáním dotazu do cache.

- Pokud existuje stav toku pro příchozí paket v cache, pak je tento paket vždy navzorkován.
- Pokud neexistuje stav toku v cache, pak je navzorkován s pravděpodobností p_v.
- V případě, že je paket navzorkován, je založen nový stav toku v cache.

Analýza tohoto algoritmu uvedená v [25] ukazuje jeho možnosti. Na příkladu níže je zachycena situace, ve které je cílem sledovat toky, které zabírají více než 1% kapacity přenosového pásma G v daném intervalu. Těchto toků nemůže být více než 100, proto cache schopná uchovat 100 stavů by měla stačit na jejich uložení, nicméně je využita paměť větší například pro 10000 stavů. Pravděpodobnost p_v navzorkování bajtu (tedy paketu obsahující příslušný bajt) je nastavena na $p_v = 10000/G$. Uvažujme tok f zabírající více než 1% kapacity. Pak tento tok posílá více než $G/100$ bajtů. Protože je každý bajt vzorkován s pravděpodobností $p_v = 10000/G$, pak pravděpodobnost, že by tok nebyl v paměti je $(1 - 10000/G)^{G/100}$, což je blízko e^{-100}. Podobně pravděpodobnost, že tok f bude v paměti již po 5% svého provozu, je $1 - e^{-5}$, což je více než 99%.

Další heuristika je založena na pravděpodobnostní struktuře o několika stupních, odtud její název Multistage Filters [25].



Obrázek 4.1: Schéma Multistage Filters.

Základní myšlenka tohoto algoritmu je znázorněna na obr. 4.1. Paket náležící toku f je namapován do několika paralelních polí pomocí několika různých hash funkcí. Každá položka pole obsahuje čítač, který je inkrementován délkou příchozího paketu. Pokud jsou všechny adresované čítače daného toku nad určeným prahem H, pak se jedná o velký tok.

Samozřejmě nastává situace, kdy se malý tok namapuje na některé čítače velkého toku. Přidáváním dalších paralelních polí se exponenciálně snižuje pravděpodobnost, že malý tok bude identifikován jako velký, neboť všechny čítače musí překročit daný práh H.

Analýza uvedená v [25] rozebírá vlastnosti tohoto algoritmu na konkrétním příkladu. Uvažujme linku s propustností 100 Mb/s se 100000 toky, kdy cílem je identifikovat toky s více než 1% provozu. Předpokládejme, že malý tok se 100 Kb/s bude označen chybně. V takovém případě musí být označen všemi paralelními poli jako velký. Každé pole je pro příklad dimenzováno pro 1000 čítačů. Aby byl 100 Kb/s tok označen jako velký, musí ostatní toky přispět do stejného čítače 900 Kb. Takových čítačů může být dle [25] nanejvýš $(100000 - 100)/900 = 111$ v jednom poli, pravděpodobnost je tedy 11%. S dalšími poli se tato pravděpodobnost násobí, tedy pro čtyři pole je pravděpodobnost špatné identifikace $1,52.10^{-4}$.

Výsledky metod vzorkování, Sample-and-Hold a Multi-stage filter jsou uvedeny v tabulce 4.1, kde nejlepším algoritmem se jeví Multi-stage filters.

Tabulka 4.1: Porovnání výsledků pro různé heuristiky (převzato z [25]).

	Neidentifikováno / Průměrná chyba		
Velikost toku	Sample-and-Hold	Multistage Filters	Vzorkování
> 0,1%	0% / 0,000008%	0% / 0,000007%	0% / 4,877%
0,1 . . . 0,01%	0% / 0,0015%	0% / 0,0014%	0,002% / 15,28%
0,01 . . . 0,001%	0,000016% / 0,1647%	0% / 0,1444%	5,717% / 39,87%

4.8 Zhodnocení

Obecně je správa paměti značně zkoumané téma. I přes značné množství publikací, které byly k tomuto tématu vydány, není možné jednoznačně říci, že určitá správa paměti je lepší než ostatní. Primárním důvodem jsou rozdílné aplikace a tedy i data, pro které jsou správy cache vyvíjeny a testovány. Dalším problémem je dostupnost použitých datových sad, což znemožňuje přesné vzájemné porovnání konkurenčních správ. V případě stejných aplikací jsou často provedena pouze srovnání nově navrhované správy paměti se základními přístupy jako jsou FIFO, LRU, LFU. V jiných případech jsou použité datové sady vytvořeny uměle. Takové sady nemusí věrně odrážet charakteristiky reálných dat a výsledky mohou být zkreslené. Nevýhodou současných správ je využití pouze některých možností optimalizace.

Vyvíjené správy jsou navrhovány s ohledem na určitou dominantní vlasnost datové sady, neboť vývojář nemůže postihnout všechny aspekty datové sady a závislosti obsažených v datech. Stejně tak není brána do úvahy velikost samotné cache, která může významně ovlivnit zaměření samotné správy.

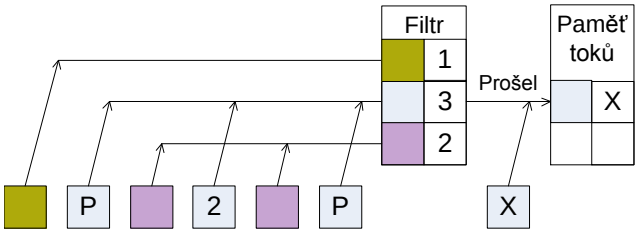
Návrh většiny správ vycházel z intuice samotných autorů a z obecného povědomí o aplikaci, ve které má být správa cache využita. Například databázové aplikace vyžadují jinou správu cache než správa instrukční cache procesoru. Správa cache překladu adres stránek virtuální paměti bude odlišná od správy cache webových stránek. Návrh správy cache na základě intuice zkušeného vývojáře může dosáhnout velmi dobrých výsledků, nicméně se zde nachází prostor pro optimalizaci správy cache, která bude kombinovat specifické vlastnosti zpracovávaných dat.

Adaptivní správy paměti založené na klasifikaci vzorů přístupů (SEQ, DEAR, UBM) lze v případě síťového provozu využít velmi obtížně. Tyto správy paměti byly navrženy pro správu cache procesorů. K takové cache přistupují řádově desítky programů, které mohou vykazovat specifické vzory přístupů k adresám do paměti. Například vykonávání smyčky programu může způsobit cyklický přístup k bloku dat. V síti se ale vyskytuje řádově tisíce instancí komunikujících aplikací. Každá instance vytvoří pouze malé množství spolu souvisejících toků. V převážné většině případů jsou vytvořeny dva toky pro přenos dat pomocí TCP. Proto u cache toků není možné tyto techniky efektivně využít.

U metody ARC a její modifikace s nastavením pomocí genetického algoritmu dochází k tomu, že na síťovém provozu konvergují k využití výhradně LRU. Vzhledem k tomu, že ARC nemá možnost vyvinout novou správu paměti, pouze balancuje mezi nastavením parametrů určité správy, zůstává otázkou, zda je LRU nejlepší možnou strategií pro správu cache síťových toků.

Optimální správu cache nelze při zpracování síťového provozu využít vzhledem k proudovému charakteru příchozích dat.

V případě využití filtrů nastává pro mnohé aplikace pracujícími se síťovými toky značný problém. Filtrovací techniky vyžadují toky nejprve klasifikovat, například formou Multis-



Obrázek 4.2: Systém pro omezení velkých P2P toků s filtrem.

tage filtru, a na základě tohoto sledování odvodit, které toky vložit do cache, a které ignorovat. Nevýhodou tohoto přístupu je ztráta počátečních paketů každého toku, a tím i závažná ztráta informace, která činí tyto techniky nepoužitelné pro aplikace využívající informaci extrahovanou z počátečních paketů toku. První pakety toku typicky obsahují ustavení spojení na aplikační vrstvě, ustavení datového přenosu na vedlejším kanálu, výměnu klíčů, přenos aplikačně specifických informací jako je například URL. Ztráta těchto informací může být pro některé aplikace sledování síťových toků nepřijatelná, neboť tuto informaci vyžadují ke svému správnému fungování. Uveďme si tuto situaci na příkladu systému pro tzv. traffic-shaping, jehož cílem je garantovat nebo omezit přenosové pásmo pro provoz náležící určité aplikaci. Za tímto účelem má systém k dispozici cache toků, ve které si udržuje záznamy o tocích. Každý záznam obsahuje identifikátor toku, stav identifikace rozpoznání aplikace a statistiky. Systém rozpoznává aplikaci na základě porovnání obsahu několika prvních paketů se sadou regulárních výrazů [29] nebo na základě vlastností prvních paketů [41]. Na základě rozpoznané aplikace systém řídí přidělování přenosového pásma danému toku. Na obrázku 4.2 je znázorněn systém pro omezení P2P toků. Velké toky jsou rozlišeny od malých pomocí Multistage filtru.

S příchodem paketů v čase se postupně plní čítače Multistage filtru. Světle modrý tok náleží P2P aplikaci a úkolem systému je tento provoz omezit, neboť tento tok přenáší velké množství nežádoucích dat (tok je velký) a zabírá dostupné přenosové pásmo ostatním aplikacím. Obrázek vystihuje situaci, kdy je tok pozdě označen filtrem jako velký a došlo tak ke ztrátě stavové informace z prvních tří paketů, která by tento tok umožnila rozpoznat jako P2P. V cache toků se vytvořil stav, který obsahuje data až ze čtvrtého paketu toku. Tato data už neposkytují žádnou informaci o aplikaci, která tok využívá. Systém tak selhává, neboť neuplatní omezení šířky pásma na tento tok. Záchranou by bylo, pokud by se informace o aplikaci vyskytovala v obsahu i dalších paketů. K tomu ale nedochází, jak můžeme odvodit z regulárních výrazů L7-filtru a dalších systémů [41]. Tato pravidla v naprosté většině případů začínají vyhrazeným symbolem pro začátek řádku, tedy začátkem toku, a proto ztráta prvního či prvních paketů je nenapravitelná.

Kapitola 5

Návrh správy cache toků

Správa cache toků může využívat některých vlastností síťového provozu daných *heavy-tail* rozložením velikostí síťových toků, shluky paketů stejného toku a limitovanou dobou trvání toku. Dále může být správa cache optimalizována na velikost paměti cache a její strukturu. Pro potřeby návrhu správy cache toků bude v následující části práce využívána terminologie ze síťové oblasti. Stav toku odpovídá jedné položce v cache, každý příchozí paket náležící danému toku znamená přístup k jeho stavu.

5.1 Definice problému

Mějme množinu toků $F = \{f_1, f_2, \ldots, f_N\}$. Optimální algoritmus je schopen dosáhnout pouze N neúspěšných vyhledání, pokud počet současně aktivních toků není větší než kapacita cache toků udávaná v počtu uložitelných stavů. V takovém případě jsou neúspěšná vyhledání způsobena pouze prvními pakety každého toku a způsobí založení stavu toku v cache. V síťových zařízeních (stejně jako u dalších reálných zařízení) není možné využít offline algoritmu pro správu cache, neboť informace o budoucím využití stavu není v době příchodu paketu k dispozici. Na řadu přicházejí algoritmy, jejichž snahou je na základě předchozích přístupů k položkám odhadnout jejich využití v budoucnosti. Označme pro tok f počet neúspěšných vyhledání, kterým by bylo možné zabránit, jako ν_f. Problém spočívá v nalezení správy S, která má co nejnižší počet výpadků stavů toků v cache, tedy:

$$\arg\min_S \sum_{f \in F} \nu_f(S), \tag{5.1}$$

kde $\nu_f(S)$ značí počet výpadků stavu toku f pro správu cache S. Počet výpadků $\nu_f(S)$ se pro různé správy cache pohybuje mezi 0 až $|f| - 1$.

Pro potřeby některých aplikací sledování stavů toků je dostatečné zaměřit se pouze na sledování stavů nejvýznamnějších toků z pohledu množství přenesených dat. Aplikace, které jsou zaměřeny na velké toky, by mohly vystačit s menší kapacitou cache než aplikace, které vyžadují sledování stavů všech toků. Doposud navržené přístupy využívají filtr, a proto nedovolují sledování velkých toků od jejich prvního paketu. V této práci navržen nový přístup ke sledování velkých toků, který spočívá v nalezení specifické správy cache toků. Hledaná správa musí upřednostňovat velké toky před malými. Toho docílí rychlou identifikací toků, které jsou neaktivní nebo nepřenáší dostatečné množství paketů. Identifikované toky jsou odstraněny z cache. Zároveň stavy toků přenášející velké množství paketů musí být chráněny před velkým množstvím malých toků.

Vyjádření tohoto problému je dáno následujícím výrazem (5.2), který hledá správu S s minimálním součtem výpadků stavů náležících do množiny velkých toků F_L:

$$\arg\min_S \sum_{f \in F_L} \nu_f(S), \tag{5.2}$$

kde $\nu_f(S)$ je počet výpadků toku f nabývající hodnoty $\nu_f = 1, \ldots, |f|$ dle zvolené správy S. Pro některé aplikace může být rovněž zajímavé vyjádřit problém tak, aby nedocházelo vůbec k výpadkům velkých toků během doby jejich aktivity, pokud je to možné.

$$\arg\min_S \sum_{f \in F_L} I(\nu_f(S)), \tag{5.3}$$

$$I(\nu_f(S)) = 1 \text{ pro } \nu_f(S) > 1,$$

$$I(\nu_f(S)) = 0 \text{ pro } \nu_f(S) = 1,$$

kde $I(\nu_f(S))$ je funkce nabývající hodnotu jedna, pokud dojde k výpadku v průběhu aktivity toku. První paket toku není uvažován, protože vždy způsobí výpadek.

5.2 Reprezentace správy cache

Úlohu správy cache definujeme nad seznamem všech stavů toků v cache. Správa cache má za úkol udržovat tento seznam seřazený dle přístupového vzoru příchodů paketů. Každý příchozí paket způsobí přístup do cache toků a zároveň spustí vykonání správy. Správa přesune stav adresovaného toku na nové místo v seznamu, pokud již v seznamu byl, nebo vloží nový stav do seznamu. V případě, že je seznam plný, dojde k odstranění posledního stavu toku v seznamu. Správa cache řadí stavy toků podle historie přístupu k těmto stavům. Tím se dostávají toky, které mají být uvolněny, na konec pomyslného seznamu. Naopak toky, které mají být uchovány, se seskupují na začátku.

Správu cache definujeme jako dvojici $S = (s, V)$, kde s je skalár (s je celé číslo z rozsahu $0 \ldots N-1$) určující pozici v seznamu, kam jsou zařazovány stavy nových toků, V je aktualizační vektor o N prvcích, kde N je délka seznamu. Vektor V se skládá z prvků $v_0, v_1, ..., v_{N-1}$, kde prvek $v_n, n = 0 \ldots N-1$, je nová pozice stavu v spravovaném seznamu. Správa paměti $S = (s, V)$ pracuje následovně.

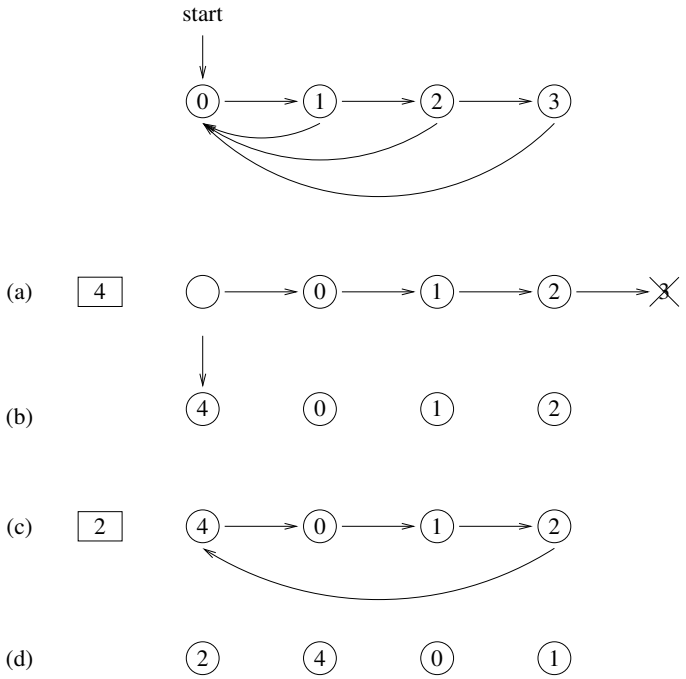
1. Pokud je stav toku f přistoupen na pozici $pos_t(f)$ v čase t (čas se inkrementálně zvyšuje s příchodem paketů), pak jeho nová pozice v seznamu je vypočítána jako $pos_{t+1}(f) = v_{pos_t(f)}$.

2. A zároveň všechny stavy mezi pozicemi $pos_{t+1}(f)$ a $pos_t(f)$ jsou posunuty o jednu pozici směrem k $pos_t(f)$.

3. Pokud stav toku f není v seznamu, pak jsou všechny stavy toku v seznamu od pozice s až ke konci seznamu posunuty o jednu pozici směrem ke konci seznamu (poslední stav v seznamu je odstraněn).

4. Nový stav je vložen na pozici s.

Správu cache S lze reprezentovat jako orientovaný graf $G_S = (U, E)$, který je sestrojen následujícím způsobem.



Obrázek 5.1: Příklad grafu správy LRU pro délku seznamu 4. Hrany se shodným počátečním i koncovým uzlem nejsou zobrazeny (například hrana $(0, 0)$ pro ponechání stavu na začátku seznamu, jednotlivé fáze jsou označeny postupně písmeny a, b, c, d).

1. Je vytvořena množina uzlů U s počtem uzlů odpovídající kapacitě seznamu, tedy N.
2. Každý uzel reprezentuje unikátní pozici toku v seznamu.
3. Uzly jsou označeny indexem $i = 0 \ldots N-1$ dle jejich pozice v seznamu, tedy $(u_0, u_1, \ldots, u_{N-1})$.
4. Množina hran E obsahuje hrany $a_0, a_1, \ldots, a_{N-1}$, kde $a_i = (u_i, u_{v_i})$ pro $i = 0 \ldots N-1$. Tyto hrany znázorňují přesun přistupovaného stavu toku f v seznamu dle hodnot prvků v_i vektoru V.
5. Dále množina hran E obsahuje hrany $b_0, b_1, \ldots, b_{N-2}$, kde $b_i = (u_i, u_{i+1})$ pro $i = 0 \ldots N-2$. Tyto hrany znázorňují stárnutí, tj. přesun stavů o jednu pozici směrem ke konci seznamu mezi uzly $u_{pos_{t+1}(f)}$ a $u_{pos_t(f)}$ nebo přesun položek při vkládání nového stavu mezi uzly u_s a u_{N-1}.
6. Do množiny uzlů U je přidán uzel *start* a do množiny E je přidána hrana $(start, u_s)$ končící v uzlu, kam jsou vkládány nové stavy toků.

Na obrázku 5.1 je ukázán graf pro příklad správy $LRU = (0, (0, 0, 0, 0))$ s délkou seznamu čtyři. Práce správy cache je znázorněna na očíslovaných stavech (kolečko) a příchozích paketech (obdelník). Shodné číslo paketu a stavu značí, že paket přísluší danému stavu (tj. uloženému toku). V situaci na obrázku 5.1 (a) je seznam plný. Při příchodu nového paketu číslo 4 je odstraněn stav číslo 3 z konce seznamu. Tím je uvolněno místo a dle LRU je $s = 0$ a nový stav toku 4 je vložen na začátek seznamu (obr. 5.1 (b)). Situace na obrázku 5.1 (c) znázorňuje příchod paketu stavu číslo 2 nacházejícího se v seznamu. Tento

stav se nachází na čtvrté pozici v seznamu a dle LRU je $v_4 = 0$ a stav 2 je přesunut na začátek seznamu. Tímto způsobem se přeskládávají stavy v seznamu dle vzoru příchodů paketů a stavy jsou tak stále seřazeny dle LRU (viz obr. 5.1 (d)).

5.3 Struktura cache toků

Dle předchozí definice správy jsou všechny stavy toků v cache umístěny v jediném seznamu, kde délka vektoru V je rovna délce tohoto seznamu. Takto velká struktura je nežádoucí, neboť pro reálné systémy, které pracují s desetitisíci toky, by správa tohoto seznamu představovala značnou režii. Tato režie by byla především spojena s nutností zajištění vyhledání a přesunů v tomto seznamu a zároveň uchováním velmi dlouhého aktualizačního vektoru V pro správu paměti.

Za účelem snížení této režie je navrženo nahrazení celého seznamu velkým množstvím kratších disjunktních seznamů $l_1, l_2, \ldots, l_M$ stejné velikosti, $\forall i, j \in 1 \ldots M : |l_i| = |l_j|$. Tyto seznamy jsou nazvány řádky, kde M je počet řádků, R je délka řádku (tj. maximální počet stavů, které může řádek obsahovat) a C velikost cache. Tyto proměnné jsou vázané vztahem:

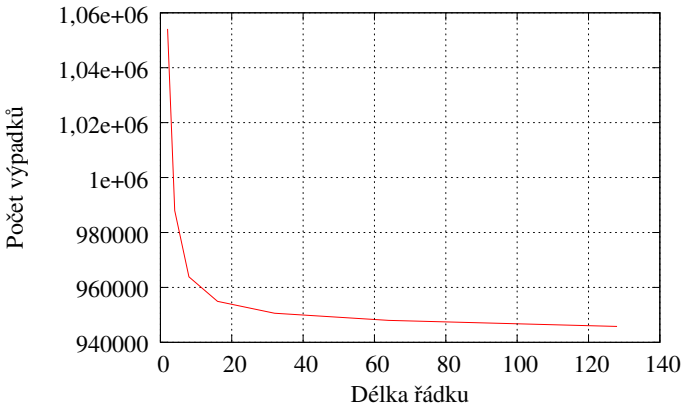
$$M = \frac{C}{R}, \tag{5.4}$$

kdy R je celočíselným dělitelem C. Všechny řádky jsou spravovány shodnou správou S ale nezávisle na ostatních. Délka vektoru V odpovídá délce řádku a není tak závislá na velikosti cache. Takto rozdělená cache odpovídá známé N-cestné cache nebo také hashovací tabulce. Vyhledání stavu toku v rozdělené cache je velmi jednoduché. Při příchodu paketu je z klíčových položek v záhlaví paketu spočítána hash. Část této hash je využita jako adresa řádku, v němž se posléze dohledá odpovídající položka porovnáním klíčových polí paketu a klíčových polí stavů toků.

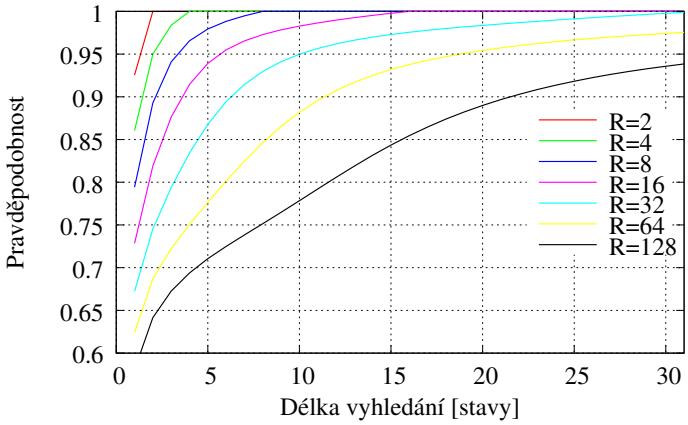
Oproti původnímu uspořádání toků v nerozděleném seznamu dochází k zanedbání části informace o pozici stavu toku. S využitím jediného seznamu obsahujícího všechny toky v cache je možné rozlišit libovolné dva stavy toků podle jejich vzdálenosti od bodu odstranění. Při využití řádků vznikají množiny stavů s ekvivalentní vzdáleností od bodů odstranění. Velikost chyby způsobena změnou uspořádání bude nižší, čím delší bude řádek. Vzniklá situace je popsána na příkladu cache spravované pomocí LRU. Mějme cache s kapacitou C toků. Postupně je prodlužována délka řádku R. Pravděpodobnost, že při plné cache je odstraněn z konce řádku stejný tok, který by byl odstraněn i v původním seznamu, je $\frac{R}{C}$. Tato pravděpobnost je velmi nízká, neboť v reálných nasazeních bude $C >> R$. Ale pro delší řádky (například délky $R > 8$) budou vybírány k odstranění takové stavy toků, které se nacházejí v blízkosti konce původního seznamu. V krajním případě bude odstraněn tok, jehož vzdálenost od konce v původním seznamu je maximálně $\frac{C}{R}$. Takový výsledek je dostačující, neboť s velkou pravděpodobností by byl tento tok odstraněn v blízké budoucnosti.

Je zřejmé, že čím bude řádek delší, tím lépe určí správa stav toku, který je vhodné z cache odstranit. Tento jev je znázorněn na obr. 5.2, který zobrazuje průběh celkového počtu výpadků ν pro zvyšující se délku řádku. Čím více toků je odstraněno, tím horší stupeň agregace daná konfigurace cache toků poskytuje, protože dochází k odstranění toků, které jsou stále aktivní.

Z obrázku je vidět, že při délce řádku $R \geq 16$ toků již nedochází k významnému snížení počtu odstraněných toků. Nemá tedy smysl délku řádku za každou cenu zvyšovat. Délka řádku je kompromisem mezi počtem prohledávaných toků při vyhledávání příslušného stavu



Obrázek 5.2: Celkový počet výpadků ν při správě LRU pro zvyšující se délku řádku R (testováno na datové sadě *Mawi-14:15* při velikosti cache pro 8192 stavů toků).



Obrázek 5.3: Kumulativní funkce pravděpodobnosti délky výhledání toku v řádku délky $R = 2, 4, 8, 16, 32, 64, 128$ se správou cache LRU.

o toku a úspěšností správy cache. Čím je delší řádek, tím více toků bude nutné prohledat a tím poroste i výpočetní náročnost vyhledání a správy. Délka řádku 32 se jeví jako dobrý kompromis mezi úspěšností výběru vhodných toků k odstranění a náročností operace vyhledání a správy cache.

Časová náročnost vyhledání by měla průměrně odpovídat polovině délky řádku. To platí pouze za předpokladu, že všechny stavy toků v řádku mají stejnou pravděpodobnost příchodu paketu. Tato situace ale v síťovém provozu nenastává, neboť každý tok má různý počet paketů, jak vypovídá graf rozložení počtu paketů v tocích na obrázku 3.2. Za těchto okolností by mělo pro vyhledání příslušného stavu v cache stačit prohledat výrazně méně než polovinu stavů v řádku. Příčinou rychlého vyhledání stavu na začátku řádku je samotná správa, z podstaty své definice řadí málo přistupované stavy na jeho konec.

Z obrázku 5.3 je patrné, že většina stavů toků, které byly úspěšně vyhledány, se nacházely na první pozici v řádku, který byl spravován pomocí LRU. Pravděpodobnost, že se tyto stavy budou nacházet na vzdálenějších pozicích, klesá exponenciálně. Tabulka 5.1 udává průměrnou pozici úspěšně vyhledaného stavu toku pro různé délky řádku.

Vyhledávání položky lze paralelizovat. Řádek v cache je možné rozdělit do několika paměťových modulů (například v FPGA) a uspořádat vnitřní strukturu stavu tak, aby

Tabulka 5.1: Průměrná délka vyhledání aktualizovaného stavu toku.

Délka řádku	2	4	8	16	32	64	128
Průměrná délka	1,07	1,21	1,44	1,91	2,81	4,63	8,28

se identifikátory toků nacházely na prvním přečteném slově, který každý modul poskytne. Toto uspořádání dovoluje současně získat několik identifikátorů, které lze paralelně porovnat s identifikátorem příchozího paketu. Pokud se nesmístí celý řádek identifikátorů do prvních slov modulů, pak na prvních místech ponecháme první část řádku a další části umístíme do slov následujících. Dle předchozích experimentů je jasné, že v první části řádku bude vyhledána většina toků a nebude nutné vyčítat identifikátory stavů toků v další části řádku.

5.4 Návrh správy cache toků pomocí genetického algoritmu

Úlohu návrhu správy cache budeme řešit pomocí genetického algoritmu. GA umožňuje postihnout širokou škálu aspektů návrhu správy cache a vzájemně je kombinovat pro dosažení lepších výsledků. Při návrhu správy cache toků jsou to především následující aspekty:

- velikost cache toků,
- velikost autonomně spravovaného seznamu toků,
- distribuce velikosti toků,
- hustota paketů v toku nebo distribuce mezipaketových mezer pro každý tok,
- průběh hustoty paketů v toku,
- typy toků,
- typ sítě,
- dynamika provozu.

Vzhledem k množství parametrů a možností návrhu správy cache je velikost stavového prostoru možných řešení obrovská a není tak možné vyzkoušet všechny možné správy. GA tedy představuje jednu z metod prohledávání stavového prostoru. GA navíc nabízí některé vlastnosti, které mohou být výhodné při nasazení správy cache v síti:

- První vlastností je oddělení prohledávaní prostoru od samotného vyhodnocení kandidátních řešení. Díky tomu není potřeba upravovat vyhledávací algoritmus při změnách požadavků aplikace, která s daty bude pracovat.
- GA využívá operace, které lze dobře realizovat v hardware s nízkou spotřebou zdrojů [60]. Typicky se jedná o bitové operace, operace sčítání a porovnání. Díky tomu je potenciálně možné realizovat vyhledávání správy přímo v hardware a urychlit tak dobu potřebnou pro vyhledání vhodné správy cache.
- GA podporuje hledání řešení i v prostředí s proměnlivými podmínkami, tj. provozem na síti. Změny síťového provozu mohou být například způsobeny nasazením nové aplikace, připojením více uživatelů. V takových případech je žádoucí, aby prohledávácí algoritmus neustále hledal vhodnou správu a zabránil zbytečnému odsunu stavů z cache toků.

- GA dokáže překonat lokální optima. Vzhledem k tomu, že existující správy představují v prohledávaném prostoru dobrá řešení, měla by tato vlastnost najít nové přístupy správy cache.

Z pohledu využití GA je nutné definovat fitnes funkci pro ohodnocení úspěšnosti vyvíjených jedinců, zakódování úlohy správy cache a určit vhodné genetické operátory, případně provést další optimalizace vedoucí k rychlejšímu prohledávání stavového prostoru. Těmto problémům jsou věnovány následující kapitoly.

5.4.1 Fitnes funkce

Různé správy dosahují odlišných výsledků ve schopnosti uchovat v cache stavy, které jsou předmětem zájmu uživatele nebo aplikace. Pro porovnání chování jednotlivých správ je vhodné navrhnout fitnes funkci, která bude hodnotit výsledky správy s hlediska nároků uživatele, programu a chování. Základním cílem správy je snížit počet výpadků ν při vyhledání stavů v cache (viz výraz (5.1)). V některých případech (například v existujícím zařízení) může být ale problematické přímo sledovat výpadky ν. Proto pro potřeby ohodnocení různých správ bude využita fitnes funkce pracující s počtem stavů toků odstraněných z cache. Počet odstraněných stavů toků z cache je označen $|F_o|$ a započítává i nevyhnutelné výpadky způsobené prvním paketem toku:

$$|F_o| = \sum_{f \in F} \nu_f(S) + |F|. \tag{5.5}$$

Pokud je znám celkový počet toků $|F|$ v datové sadě, pak je možné dopočítat i počet výpadků a rovněž pravděpodobnost výpadků p_ν. K výpočtu jsou využity výsledné stavy o tocích, které jsou zaznamenány při odstranění stavů toků z cache. V průběhu a po skončení měření je akumulován $|F_o|$ (na konci měření dojde k vyprázdnění cache). Zároveň je sledován celkový počet paketů $\sum_{f \in F} |f|$, který lze vypočítat součtem hodnot čítače paketů z každého odstraněného stavu. Pravděpodobnost nevyhnutelných výpadků p_ν lze následně vypočítat jako poměr:

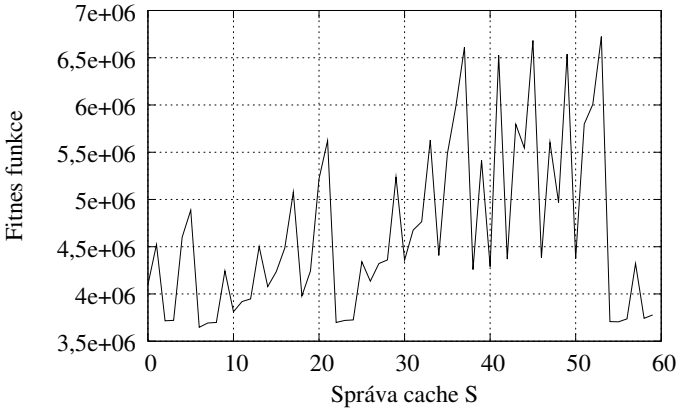
$$p_\nu = \frac{|F_o| - |F|}{\sum_{f \in F} |f| - |F|}. \tag{5.6}$$

Pokud je správa ideální, pak $|F_o|$ je rovno $|F|$ a pravděpodobnost výpadků $p_\nu = 0$. Pokud správa naprosto selže a pro každý paket odstraní tok z cache, pak $|F_o|$ je rovno $\sum_{f \in F} |f|$ a $p_\nu = 1$. Vzorec záměrně nezapočítává výpadek pro první paket toku, protože tento výpadek je nevyhnutelný. Pro účely vývoje správy s co nejnižším celkovým počtem výpadků, je jako fitnes funkce využit přímo výraz (5.5). GA se tedy snaží snížit počet odstraněných toků z cache.

Obrázek 5.4 poskytuje náhled na výřez průběhu uvedené fitnes funkce. Tato funkce každé správě (délky řádku $R = 4$) přiřazuje počet odstraněných stavů toků z cache. Správa je reprezentována dle sekce 5.2 skalárem s a vektorem V, kde tyto položky postupně nabývají všech přípustných variací s opakováním. Čím nižší je hodnota fitnes funkce[1], tím úspěšnější je správa.

Ukázka na obr. 5.4 zároveň dává představu o chování této funkce. Je vidět, že průběh má mnoho lokálních minim, ve kterých by jednoduchý prohledávácí algoritmus mohl uváznout.

[1] U GA je zvykem, že vyšší hodnota fitnes funkce značí lepší řešení. Vzhledem k tomu, že v této práci jsou počítány výpadky, bylo by nutné zavést další funkci pro úravu výsledků dle zavedeného zvyku. Pro účely této práce nebude výsledek ohodnocení upravován, neboť výsledek přímo vypovídá o úspěšnosti správy cache.



Obrázek 5.4: Výřez fitnes funkce pro problém malého rozsahu měřeného na části vzorku *Mawi-2010/04/14-14:00* s velikostí cache pro 8192 stavů.

Zároveň se dvě sousední řešení mohou významně lišit svou úspěšností. Systematické ohodnocení všech řešení pro větší délky řádku je časově příliš náročné. Tato náročnost je způsobena tím, že daná správa cache se musí ohodnotit na reálném vzorku dat, který nesmí být příliš krátký, jinak by pozbyl svou vypovídající hodnotu (vhodná délka vzorku je diskutována v sekci 5.5).

Dalším řešeným problémem je snížení počtu výpadků stavů pouze u velkých toků (viz (5.2)). Výpočet fitnes funkce zachycující tento problém je náročnější než výpočet fitnes funkce zachycující problém snížení celkového počtu výpadků pro všechny toky. Je nutné sledovat nejen celkový počet výpadků, ale i počet výpadků pro každý tok a navíc každý tok po jeho konci přiřadit do správné kategorie. Při simulaci jsou odstraněné stavy uchovány v seznamu. Následný průchod tímto seznamem spojuje stavy náležící stejným tokům. Vznikají tak stavy, které odráží průběh celého toku (především jeho začátek, konec, počet paketů, bajtů). Zároveň je v těchto stavech uložen počet výpadků odvozených z počtu odstraněných stavů stejného toku. Z výsledných stavů je spočítána penalizovaná šířka pásma r_f toku f dle výrazu (3.1). Na základě výsledku je tok přiřazen do množiny velkých toků F_L nebo je ignorován, neboť je malý. Pravděpobnost výpadků stavů velkých toků $p_{\nu_{F_L}}$ lze následně vypočítat jako:

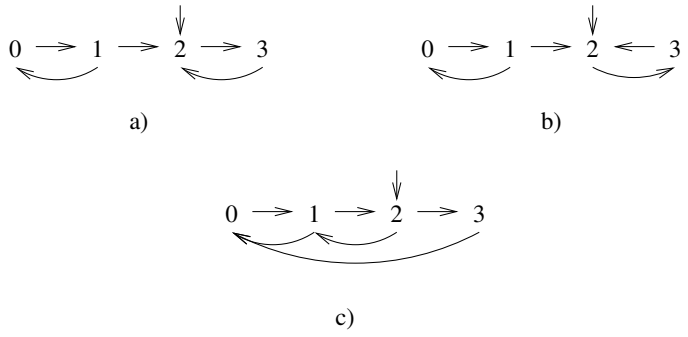
$$p_{\nu_{F_L}} = \frac{\sum_{f \in F_L} \nu_f(S)}{\sum_{f \in F_L} |f| - |F_L|}.$$

Za účelem vývoje správy s co nejnižším počtem výpadků u velkých toků je využit následující výraz:

$$100 \times (|L_1| + \sum_{f \in F_{L_1}} \nu_f(S)) + 10 \times (|L_2| + \sum_{f \in F_{L_2}} \nu_f(S)) + |L_3| + \sum_{f \in F_{L_3}} \nu_f(S). \qquad (5.7)$$

Ohodnocení správy tak odráží kromě výpadků také jemnější rozdělení toků dle jejich velikosti do několika kategorií. Výsledná správa cache je hodnocena jako váhovaný součet odstraněných stavů toků v prvních třech kategoriích velkých toků. Výpadky toků s nejvyšším využitím linky mají nejvyšší váhu, pro ostatní se váha snižuje. Váha kategorií je zvolena tak, aby odrážela poměry, ve kterých byly vymezeny jednotlivé kategorie. Tedy L_1 toky jsou desetkrát intenzivnější než toky v L_2 atd.

Úspěšnost správy lze hodnotit také na základě sumy nepřerušených velkých toků dle



Obrázek 5.5: Příklady správ – a) $S = (2, (0, 0, 2, 2))$, b) $S = (2, (0, 0, 3, 1))$, c) $S = (2, (0, 0, 1, 0))$.

výrazu (5.3). Fitnes funkce pak odpovídá následujícímu výrazu:

$$100 \times (|L_1| + \sum_{f \in F_{L_1}} I(\nu_f(S))) + 10 \times (|L_2| + \sum_{f \in F_{L_2}} I(\nu_f(S))) + |L_3| \sum_{f \in F_{L_3}} I(\nu_f(S)). \quad (5.8)$$

Nicméně lze očekávat, že tato funkce povede na vývoj velmi podobné správy paměti jako u předchozí výrazu (5.7).

5.4.2 Zakódování úlohy správy cache

Pro zakódování správy cache využijeme přímo reprezentaci správy ze sekce 5.2, která správu S definuje jako dvojici skaláru s (pozice v seznamu, kam jsou zařazovány stavy nových toků) a aktualizačního vektoru V. Rozsah hodnot, který může každý prvek nabývat, je závislý na délce spravovaného seznamu. Ve zbytku práce bude využíváno rozčlenění seznamu do mnoha nezávisle spravovaných řádků popsané v předchozí sekci 5.3. Proto rozsah hodnot prvků v chromozomu nabývá hodnot $0 \ldots R-1$, kde R je délka řádku odpovídající délce vektoru V.

Velikost prostoru řešení závisí tedy na délce R, počtu možností, kam přesunout z jakékoliv pozice stav (tj. R možností), a počtu možností kam vložit nový stav (opět R možností). Velikost prohledávaného prostoru obsahuje R^{R+1} různých řešení. Pro délku řádku $R = 32$ existuje $4,7.10^{49}$ možných řešení. Vzhledem k velikosti prohledávaného prostoru je vhodné provést optimalizace s cílem zkrátit dobu GA potřebnou pro nalezení úspěšného řešení.

Zásadní optimalizací je omezení prohledávaného stavového prostoru. Na obr. 5.4 je možné vidět, že některé správy mají výrazně horší výsledky než jiné. Pokud se podaří tyto špatně fungující správy odstranit ještě před jejich ohodnocením, pak se uspoří významné množství výpočetního času. Především v prvních iteracích GA je velká pravděpodobnost, že náhodně vygenerovaní jedinci budou mít špatné ohodnocení. Včasné odhalení a odstranění těchto jedinců je založeno na několika úvahách.

Optimalizace Opt_1. Správa, jenž nevyužívá celou kapacitu řádku (například obrázek 5.5 situace a), dosáhne horších nebo stejných výsledků jako správa z ní vychazející a využívající všech pozic v řádku, tj. jeho plnou kapacitu. Správy nevyužívající plnou kapacitu řádku je vhodné upravit. Postup úpravy vychází z následujícího zdůvodnění.

Zdůvodnění Opt_1. Nechť existuje $S = (s, V)$, která nevyužívá celou kapacitu řádku. Dle návodu v sekci 5.2 je sestaven orientovaný graf G odpovídající správě S. Pokud S nevyužívá plnou kapacitu řádku, pak některé uzly v G_S nebudou dosažitelné z uzlu *start*. Jakýkoliv takto sestavený graf je vždy alespoň slabě souvislý díky hranám b. Pro všechny uzly u_i,

kde $i \geq s$ platí, že jsou vždy dosažitelné z uzlu *start*. Graf G je upraven na graf G' tak, že jedna z hran a_i, kde $i \geq s$, je změněna na $a_i = (u_i, u_0)$. Všechny uzly v G' jsou dosažitelné z uzlu *start*. Správa S_{Opt_1} reprezentovaná grafem G' musí být stejně nebo více úspěšná jako správa S, neboť S_{Opt_1} přesune stav z pozice u_i do původně nevyužívané části grafu. Tím vytvoří volné místo pro další stav, který by nebylo možné dříve uložit bez odsunu jiného stavu. Pravděpodobnost výpadku stavu v cache je tak stejná nebo nižší.

Optimalizace Opt_2. Správa, která zhorší pozici aktualizovaného stavu, tj. přesune stav blíže konci řádku (například obrázek 5.5b), dosáhne horších výsledků než správa cache, která nezmění nebo zlepší pozici aktualizovaného stavu. Správy zhoršující pozici aktualizovaného stavu je vhodné upravit. Postup úpravy vychází z následujícího zdůvodnění.

Zdůvodnění Opt_2. Nechť existuje správa S, která zhorší pozici aktualizovaného toku f, tedy $\exists v_i \in V : v_i \geq i$. Při tomto přesunu se zlepší pozice neaktualizovaných stavů nacházejících se mezi starou a novou pozicí f. Správa S je upravena na S_{Opt_2} tak, že alespoň jedna položka v_i vektoru V, pro kterou platí $v_i \geq i$, je upravena na $v_i = i$. Správa cache S_{Opt_2} musí dávat shodné nebo lepší výsledky než S, neboť v síťovém provozu je zachován princip lokality. Tedy je pravděpodobnější, že aktualizovaný stav bude aktualizován i v blízké budoucnosti. Nicméně pokud správa S přesouvá aktualizované stavy blíže konci seznamu, pak se zvyšuje pravděpodobnost jejich odstranění z cache, tj. zvyšuje se pravděpodobnost výpadků.

Optimalizace Opt_3. Správa, která nezlepší pozici aktualizovaného stavu, tj. přesune stav blíže konci řádku nebo ponechá stav na své původní pozici, dosáhne horších výsledků než správa, která zlepší pozici aktualizovaného stavu. Správy nezlepšující pozici aktualizovaného stavu je vhodné upravit. Postup úpravy vychází z následujícího zdůvodnění.

Zdůvodnění Opt_3. Tato optimalizace v sobě zahrnuje předchozí dvě optimalizace. Opt_3 je založena na myšlence, že pokud správa S ponechá při aktualizaci stav na původní pozici, pak ztrácí informaci o provedeném přístupu k stavu a nemůže být tak přesná jako správa $S_{Opt_3} = (s, V')$, která vznikne následující modifikací S:

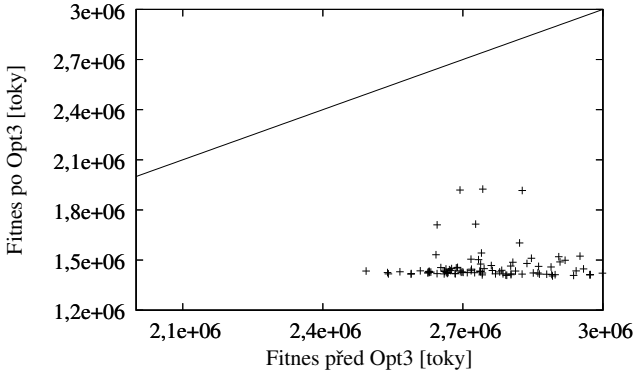
$$\forall v'_i \in V' : v'_i = \begin{cases} v_i & \text{pro } v_i \leq i \\ i - 1 & \text{jinak.} \end{cases} \tag{5.9}$$

Zavedením optimalizace Opt_3 se zmenšila velikost prohledávaného stavového prostoru. Nový prostor respektující optimalizaci modifikací dle výrazu (5.9) obsahuje $R(R-2)!$ různých řešení, neboť je omezen počet možností, kam lze přesouvat aktualizované stavy. Například pro řádek délky 32 to znamená omezení prostoru z $1,46.10^{48}$ na $2,63.10^{35}$ řešení.

Před tím než bude optimalizace Opt_3 využita v GA, je vhodné experimentálně ověřit, zda tato optimalizace neodstranila některá slibná řešení správy cache. Náhodně jsou vygenerovány správy S, na něž jsou provedeny modifikace dle výrazu (5.9). Výsledkem je S_{Opt_3}. S i S_{Opt_3} jsou ohodnoceny z hlediska počtu odstraněných toků v simulaci. Cílem je zjistit, zda dojde ke zlepšení či zhoršení výsledku S_{Opt_3} vůči původní S.

Náhodně je vygenerováno 1000 instancí správ a každá z nich je ohodnocena simulací nad zkráceným vzorkem dat *Mawi-2010/04/14-14:00* s 12 miliony pakety. Následně jsou provedeny modifikace těchto správ optimalizací Opt_3 dle předpisu (5.9). Pro každou dvojici původní a optimalizované správy je zaznačen bod do grafu na obrázku 5.6, kde x-ová souřadnice bodu odpovídá ohodnocení správy S a y-ová souřadnice ohodpovídá odnocení správy S_{Opt_3}.

Lze pozorovat, že všechny body se nacházejí pod přímkou $y = x$. S_{Opt_3} je tedy vždy hodnocena lépe, v převážné většině případů je zlepšení až dvojnásobné. Na základě tohoto výsledku lze usuzovat, že optimalizované S_{Opt_3} správy cache dosáhují lepších výsledků než



Obrázek 5.6: Korelace výsledků správ cache toků bez optimalizace a po optimalizaci Opt_3 (pro přehlednost je zobrazeno prvních 100 vzorků, které tvoří reprezentativní zobrazení).

původní S. Zároveň je pouze velmi malá pravděpodobnost opačného jevu. Optimalizaci Opt_3 lze tedy využít pro redukci stavového prostoru možných řešení s velmi nepatrným rizikem odstranění potenciálně slibné správy cache.

Optimalizace Opt_4. Správa S_{Opt_3}, která dovolí přeskakování při přesunech stavů (například obrázek 5.5c) z lepší pozice stavem z horší pozice, dosáhne horších výsledků než správa, která toto přeskakování neobsahuje. Správa cache obsahující přeskakování nesplňuje následující podmínku (5.10):

$$\forall v \in V : v_i \leq v_j, \text{pro } i \leq j. \tag{5.10}$$

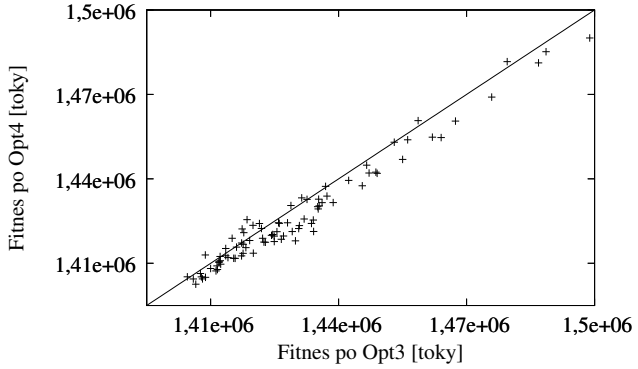
Správy dovolující přeskakování je vhodné upravit. Postup úpravy vychází z následujícího zdůvodnění.

Zdůvodnění Opt_4. Mějme správu S_{Opt_3}, která obsahuje přeskakování. Modifikujeme správu $S_{Opt_3} = (s, V)$ na $S_{Opt_4} = (s, V')$ následovně:

1. Ve vektoru V vyhledáme v_i v_j, kde $v_i \leq v_j \wedge i \geq j$, a zároveň i a j jsou nejvyšší indexy splňující tuto podmínku,
2. Vzájemně prvky ve vektoru vyměníme. $S_{Opt_4} = (s, V')$, kde $v'_k = v_k$ a $v'_i = v_j, v'_j = v_i$.

Můžeme předpokládat, že správa S_{Opt_4} dává stejné nebo lepší výsledky než S_{Opt_3}. Je pravděpodobné, že stav, který se vyskytoval na horším místě v seznamu, bude náležet toku, jenž má špatnou lokalitu. Jeho přesun před aktualizovaný stav z lepší pozice zvyšuje riziko odsunu stavu s vyšší lokalitou. Před nasazením optimalizace Opt_4 je nutné experimentálně ověřit, zda platí hypotéza o špatném vlivu přeskakování či nikoliv. Při zavedení Opt_4 by se výrazně snížila velikost prohledávaného prostoru, protože vektor V správ cache splňující podmínku 5.10 bude vždy neklesající posloupnost hodnot. Počet možných řešení U_R, kde R je délka řádku, tak můžeme vyčíslit rekurzivně jako:

$$\begin{aligned}
Suma(1, j) &= 1, && (5.11)\\
Suma(i, j) &= 0, \text{pro } j \geq i, && (5.12)\\
Suma(i, j) &= Suma(i-1, j) + Suma(i-1, j-1) + \ldots && (5.13)\\
&\quad \ldots + Suma(i-1, 1), \text{pro } j < i, && (5.14)\\
U_R &= R \cdot Suma(R+1, R-1), && (5.15)
\end{aligned}$$



Obrázek 5.7: Korelace výsledků správ cache po optimalizaci Opt_3 a po následné optimalizaci Opt_4 (pro přehlednost je zobrazeno prvních 100 vzorků, které tvoří reprezentativní zobrazení).

kde $i = 1 \ldots R, j = 0 \ldots R-1$ a $Suma(i, j)$ vyjadřuje počet možných neklesajících posloupností délky i končících prvkem s hodnotou j splňující podmínku (5.10). $Suma(R+1, R-1)$ vyjadřuje součet všech hodnot v předchozí iteraci R a tedy i celkový počet možných neklesajících posloupností. Pro získání celého univerza ještě musíme tyto možnosti pronásobit délkou řádku R, což vyjadřuje počet možností vložení nového stavu do řádku. Vyčíslení několika iterací výrazu (5.15) je uvedeno v tabulce 5.4.2 pro délku řádku $R = 4$. Vyčíslení iterací pro $R = 32$ ukazuje snížení velikosti prostoru řešení na $1,2.10^{17}$, což je velmi významná úspora. Pro řádek délky 32 je $U_{32} = 1,78.10^{18}$, redukce stavového prostoru o 29 řádů.

Tabulka 5.2: Vyčíslení sumy neklesajících posloupností pro $R = 4$ ($U_4 = 4 \cdot Suma(5, 4) = 56$).

Suma	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$j = 0$	1	1	1	1	1
$j = 1$	0	1	2	3	4
$j = 2$	0	0	2	5	9
$j = 3$	0	0	0	5	14

Důsledky optimalizace Opt_4 na správu cache jsou experimentálně ověřeny. Na stávající Opt_3 optimalizované správy cache je aplikována Opt_4. Touto operací vzniknou nové správy, které odpovídají oběma optimalizacím Opt_3 i Opt_4.

Nové správy S_{Opt_4} jsou ohodnoceny a výsledky vyneseny jako body do grafu zobrazeného na obrázku 5.7. Souřadnice x odpovídá ohodnocení před Opt_4 optimalizací a souřadnice y ohodnocení po optimalizaci. Dle polohy umístění bodů lze usuzovat, zda dojde ke zlepšení nebo zhoršení výsledků po optimalizaci. V grafu je vidět, že většina (kolem 80%) bodů se nachází pod přímkou $y = x$ nebo v její těsné blízkosti. To znamená, že ve většině případů došlo ke zlepšení nebo zachování výsledků správy cache po optimalizaci. V některých případech došlo ovšem ke zhoršení výsledku. Je tedy jasné, že touto optimalizací bychom mohli z prohledávaného univerza odstranit správy cache, které dosahují lepších výsledků.

Optimalizace Opt_5. Správu je možné překódovat do úspornější reprezentace při ztrátě

určité informace. Změna správy S spočívá ve změně reprezentace jejího aktualizačního vektoru V do zakódování úseků. Místo vektoru V je zaveden vektor dvojic $W = ((c_0, w_0), \ldots (c_{Z-1}, w_{Z-1}))$, kde c_j představuje počet sousedních prvků se stejnou hodnotou w_j v původním vektoru V pro $j = 0 \ldots Z-1$. Z je počet souvislých úseků se stejnou hodnotou w_j. Například vektor $V = (0, 0, 1, 1, 1)$ lze reprezentovat jako $W = ((2, 0), (3, 1))$. Zároveň platí, že $\sum_{j=0\ldots Z-1} c_j = R$, kde R je délka řádku. Pro zachování Opt_3 mohou hodnoty w_j nabývat hodnot $0 \ldots -1 + \sum_{k=0\ldots j-1} c_k$. Jakýkoliv vektor W je možné převést na vektor V dle následujícího postupu. Pro $i = 0 \ldots R-1$:

$$v_i = w_0 \text{ pro } 0 \leq i < c_1, \tag{5.16}$$

$$v_i = w_j \text{ pro } \sum_{k=0\ldots j-1} c_k \leq i < (\sum_{k=0\ldots j} c_k). \tag{5.17}$$

Za účelem snížení velikosti stavového prostoru je nutné, aby počet souvislých úseků Z byl násobně menší než délka řádku R. Tím dojde ke sloučení informace o přesunech stavů z několika sousedních pozic do jedné hodnoty. Pokud zvolíme například $Z = 8$, pak průměrně na každé c_j připadají 4 prvky v nezakódovaném vektoru V. Každé c_j může nabývat hodnot v rozsahu $1 \ldots 7$ při zachování podmínky:

$$\sum_{j=0\ldots Z-1} c_j = 32. \tag{5.18}$$

Velikost prohledávaného prostoru závisí na počtu kombinací přípustných hodnot prvků c_j a možných kombinací hodnot prvků w_j. Jednoduchým programem lze vyčíslit všechny možné kombinace $c_0, \ldots c_7$, a započítáme ty, jejichž součet je právě 32. Výsledek je vynásoben počtem možných kombinací hodnot prvků w, kdy $w_1 = 0, w_1 = \langle 0, 7\rangle, w_2 = \langle 0, 15\rangle, w_3 = \langle 0, 23\rangle, w_{4\ldots 7} = \langle 0, 31\rangle$. Horní odhad velikosti celého stavového prostoru pro $Z = 8$ je $1,3.10^{15}$. Skutečný počet řešení by musel vzít v úvahu některé nepřípustné kombinace hodnot prvků c a w. K ohodnocení každého jedince je dle výrazů (5.16), (5.17) rozgenerován příslušný vektor W na V. Vektor V je následně ohodnocen pomocí simulace správy cache nad vzorkem dat.

Optimalizace Opt_6. Rozšíření zakódovaní úseků dovoluje reprezentovat změny v rámci úseku a obnovuje tak ztracenou informaci v reprezentaci Opt_5. Reprezentace vektoru V je rozšířena pomocí vektoru trojic. Místo vektoru V je zaveden vektor trojic $Q = ((c_0, w_0, q_0), \ldots (c_{Z-1}, w_{Z-1}, q_{Z-1}))$, kde c_j představuje počet sousedních prvků v posloupnosti, w_j představuje hodnotu prvního prvku posloupnosti a q_j celočíselný přírůstek hodnot sousedních prvků posloupnosti, Z je počet posloupností. Například vektor $V = (0, 0, 0, 1, 2, 3)$ lze reprezentovat jako $Q = ((2, 0, 0), (4, 0, 1))$. Stále platí, že $\sum_{j=0\ldots Z-1} c_j = R$, kde R je délka řádku. Pro zachování Opt_3 mohou hodnoty w_j nabývat hodnot $0 \ldots \sum_{k=1\ldots j-1} c_k$ pro $q_j \leq 1$. To znamená, že prvky v posloupnosti mohou zůstat stejné jako první prvek nebo růst či klesat postupně o jedna. Jakýkoliv vektor Q je možné převést na vektor V dle následujícího postupu. Pro $i = 0 \ldots R-1$:

$$v_i = w_0 + i * q_0 \text{ pro } 0 \leq i < c_1, \tag{5.19}$$

$$v_i = w_j + (i - \sum_{k=0\ldots j-1} c_k) * q_j \text{ pro } \sum_{k=0\ldots j-1} c_k \leq i < \sum_{k=0\ldots j} c_k. \tag{5.20}$$

Pro délku řádku $R = 32$ je zvolen počet posloupností $Z = 8$. Počet posloupností je možné zvýšit či snížit na základě výsledků experimentů. Velikost prohledávaného prostoru

se tak zvýší 6561-krát díky přidání osmi prvků q_j, které nabývají tří možných hodnot. Horní odhad velikosti stavového prostoru je $3,3.10^{20}$. K ohodnocení každého jedince je dle výrazů (5.19), (5.20) rozgenerován příslušný vektor Q na V, který je následně ohodnocen pomocí simulace správy cache nad vzorkem dat.

5.4.3 Genetické operátory

Hlavním motorem změny jedinců je v uvažovaném algoritmu operátor mutace. Snahou operátoru mutace je zvýšit diverzitu nově vznikající populace a nacházet nová řešení. Operátor mutace musí respektovat dané zakódování jedince. Nicméně pro dané zakódování může být navrženo více operátorů. Genetické operátory budou definovány pouze pro optimalizace Opt_3, Opt_4, Opt_5, Opt_6. Optimalizace Opt_1 a Opt_2 jsou již zahrnuty v Opt_3.

V souladu s optimalizací Opt_3 definujeme základní operátor mutace O_{mut}. Tento operátor s pravděpodobností p_{mut} nahrazuje každý prvek vektoru V tak, že vygeneruje náhodné číslo z intervalu $0 \dots R-1$ a nahradí jím tento prvek vektoru V. Do tohoto operátoru je zavedena optimalizace Opt_3 následujícím způsobem. Pokud probíhá mutace daného prvku, pak rozsah generovaného náhodného čísla je omezen na rozmezí $0 \dots k-1$, kde k je pozice prvku v_k ve vektoru V a k nabývá hodnot $1 \dots R-1$. Prvek na pozici 0 nabývá vždy hodnotu 0. Pozice vkládání s nových stavů může mutovat libovolně mezi hodnotami $0 \dots R-1$.

Při optimalizaci Opt_3 nabývají prvky vektoru V se zvyšujícím se indexem vyššího počtu hodnot. Z tohoto pohledu se jeví vhodné snížit pravděpodobnost mutace prvků s malým rozsahem hodnot a naopak zvýšit pravděpodobnost prvkům na konci vektoru. Tato úprava by měla vést k lepšímu prohledávání univerza tím, že budou generována rozmanitější řešení. Mutační operátor O_{mut} rozšíříme o proměnnou pravděpodobností mutace $p_{mut}(v)$. Pravděpodobnost mutace roste lineárně se zvyšujícím se indexem prvku v vektoru V.

Dále zavedeme operátor mutace O_{mut-1}, který s pravděpodobností $p_{mut}(v)$ změní prvek vektoru V tak, že přičte či odečte jedničku k nebo od stávající hodnoty prvku. Zároveň tento operátor implementuje optimalizaci Opt_3 omezením rozsahu hodnot prvku dle jeho pozice. Pokud se nachází výsledné číslo mimo rozsah, pak je navráceno na nejbližší hraniční hodnotu, tedy upravený prvek v'_i je:

$$v'_i = min(max(v_i, 0), i-1),$$

kde 0 je spodní hranice a $i = 1 \dots R-1$ horní hranice i-tého prvku vektoru. Je zřejmé, že by bylo možné při přesáhnutí rozsahu přejít na hodnotu vzdálenější hranice, aby všechny hodnoty měly své dva sousedy. Na druhou stranu sémantika obou hraničních hodnot je opačná a nejeví se tak vhodné tyto hodnoty pokládat za sousední.

Operátor mutace O_{mut} s proměnnou pravděpodobností mutace $p_{mut}(v)$ je zaveden i pro optimalizaci Opt_4. Opt_4 je implementována tak, že po mutaci daného prvku v_x s indexem x je spuštěna úprava vektoru. Tato úprava změní nebo ponechá původní hodnoty prvkům dle následujícího předpisu:

$$v'_i = min(v_i, v_x), i \leq x,$$

$$v'_i = max(v_i, v_x), i > x,$$

kde $i = 0 \dots R-1$ a v'_i jsou prvky upraveného vektoru.

Pro Opt_5 je implementován mutační operátor O_{mut-w}. Tento operátor aplikuje O_{mut} nad vektorem dvojic W, kde s pravděpodobností p_{mut} jsou pozměněny prvky c_j a w_j a prvek s. Po aplikování operátoru se spustí funkce *fixc()*, která upraví vygenerované jedince na validní. Operátor mutace O_{mut-w} je popsán následujícím pseudokódem:

```
R ← 32, Z ← 8, MAXVAL ← 7
{C-MUTATE}
for i = 0 ... Z − 1 do
  if random() ≤ p_mut then
    c_i ← randint(1, MAXVAL)
  end if
end for
fixc(c, R)

{W-MUTATE}
maxmove ← 0
for i = 1 ... Z do
  if random() ≤ p_mut then
    w_i ← randint(1, maxmove)
  end if
  if c_i ≥ maxmove then
    w_i ← maxmove
  end if
  maxmove ← maxmove + c_i
end for

{S-MUTATE}
if random() ≤ p_mut then
  s ← randint(0, R − 1)
end if

{FIX-C}
fixc(c, R):
c_diff ← sum(c) − R
inc ← (c_diff > 0)?1 : (−1)
while c_diff <> 0 do
  randindex ← randint(0, Z − 1)
  tmpc ← c_randindex − inc
  if tmpc ∈ {1 ... MAXVAL} then
    c_randindex ← tmpc
    c_diff ← c_diff − inc
  end if
end while
```

Smyčka označená návěstím C-MUTATE má za úkol s pravděpodobností p_{mut} nahradit prvky ve vektoru c. Funkce $random()$ generuje náhodná reálná čísla z rozsahu $\langle 0, 1)$. Funkce $randint()$ generuje náhodná celá čísla z rozsahu uvedeného v parametrech funkce. Po mutaci některých prvků vektoru c dojde k porušení podmínky (5.18). V takovém případě musí být prvky vektor c upraven, aby součet jeho prvků odpovídal délce řádku. Za tímto účelem je spuštěna procedura *fixc()*, jejíž implementace je popsána za návěštím FIXC. Nápravy jedince je dosaženo opakovaným zvýšením/snížením hodnoty náhodného prvku vektoru c o jedna, dokud není vyrovnán rozdíl mezi sumou prvků vektoru c a délky řádku na nula. Smyčka s návěstím W-MUTATE s pravděpodobností p_{mut} nahradí hodnotu prvku w v od-

povídajím rozsahu, aby byly dodrženy podmínky Opt_3. Nakonec dojde s pravděpodobností p_{mut} i k mutaci skaláru určujícího místo vkládání nových záznamů (návěstí S-MUTATE).

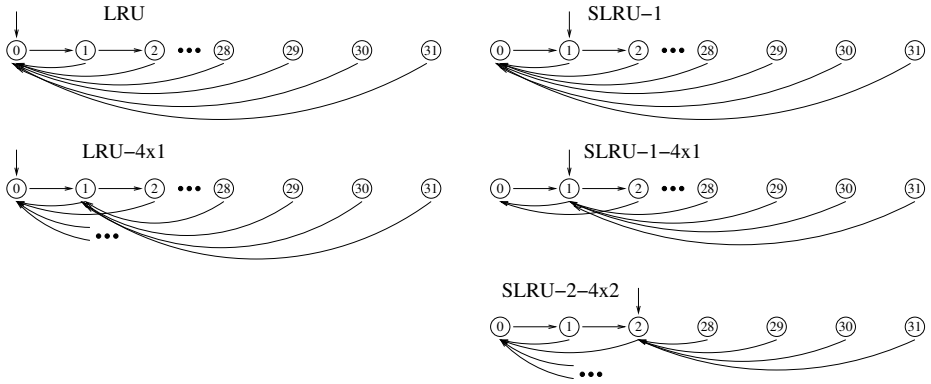
Pro optimalizaci Opt_6 je rozšířen O_{mut-w} na O_{mut-q}, který aplikuje s pravděpodobností p_{mut} mutaci O_{mut} na všechny typy prvků vektoru Q. Po mutaci následuje oprava chromozomu *fixc()*. Stojí za povšimnutí, že první prvek každého úseku i nabývá hodnotu w_i, která je generována tak, aby byla zachována optimalizace Opt_3. Zároveň prvky q_i inkrementující hodnoty w_i mohou nabývat maximálně hodnotu jedna. Optimalizace Opt_3 tak přímo splněna pro všechny prvky v úseku, neboť pokud ji splňuje první prvek, pak musí i další prvky v úseku.

U genetického algoritmu se operátor křížení stará o výměnu genetické informace mezi jedinci. Z pohledu vývoje správy cache může křížení zajistit výměnu úseků vektoru V. Lze ale očekávat, že křížení nebude mít velký přínos při hledání správy cache, neboť vzájemně se jednotlivé části vektoru ovlivňují, tj. v průběhu vývoje vznikají vzájemné vazby částí vektoru. Tyto vazby budou ve většině případů křížením zpřetrhány, a proto výsledný jedinec s velkou pravděpodobností zanikne. Z tohoto důvodu je křížení definováno pouze pro zákódování Opt_5 a Opt_6.

Jako první je zavedeno křížení, při kterém dojde k vzájemné výměně prvků (c_i, w_i) na pozici i ve vybraných vektorech W_1 a W_2 s pravděpodobností p_{cross}. Toto křížení je označeno $O_{cross-d}$. Dále je zavedeno jednobodové křížení $O_{cross-b}$, při kterém dojde k rozdělení W_1 a W_2 na náhodně zvolené pozici a dojde k vzájemné výměně rozdělených částí vektoru s pravděpodobností p_{cross}. U obou křížení může dojít k porušení podmínky (5.18). Pro navrácení nově vzniklých vektorů do prostoru přípustných řešení je, stejně jako po mutaci, po křížení spuštěna procedůra *fixc()*.

5.5 Zkrácení doby ohodnocení

Zkrácením datové sady lze zkrátit dobu simulace a tím celkově přispět k hledání úspěšné správy cache. Každou správu cache, která je navržena pomocí GA, je třeba ohodnotit. Doba ohodnocení navržených řešení zabírá nejvíce výpočetního času běhu GA. Na délce trvání ohodnocení řešení se podílí mimo jiné velikost datové sady, velikost cache toků, délka řádku. Umístění simulované cache do hlavní paměti počítače nebo do cache procesoru záleží na její velikosti. V případě, že velikost cache toků je násobně menší než velikost cache procesoru, pak se celá cache toků smístí do cache procesoru a přístupy k stavům jsou velmi rychlé, neboť nedochází k výpadkům. Pokud velikost cache toků přesahuje velikost cache procesoru, pak dochází k výpadkům a doba potřebná pro ohodnocení správy je delší. Nicméně velikost cache toků není možné ovlivnit, neboť je jedním z parametrů, pro který GA vyvíjí optimalizovanou správu cache. Rovněž délka řádku může částečně ovlivnit délku doby ohodnocení. Ze stejného důvodu jako velikost cache toků ji ale není možné změnit. Posledním faktorem, který je možné optimalizovat, je velikost vzorku dat využitých při simulaci. Zkrácením vzorku dat dojde ke úměrnému zkrácení doby ohodnocení. Zkrácení ovšem nesmí ovlivnit kvalitu ohodnocení správy cache. Pokud by se tak stalo, vyhledávací algoritmus by dostával špatné údaje pro prohledávání stavového prostoru a snížila by se pravděpodobnost nalezení kvalitní správy. Je proto vhodné zachovat následující pravidlo. Pro všechny různé správy cache S_1 a S_2, jejichž ohodnocení v simulaci je $o(S_1, D)$ a $o(S_2, D)$, kde D značí vzorek dat, platí následující podmínka. Pokud je vzorek dat redukován (například zkrácením na polovinu) a označen D_z, pak ohodnocení $o(S_1, D_z)$ a $o(S_2, D_z)$ musí být ve stejném poměru jako $o(S_1, D)$ a $o(S_2, D)$. V nejzasším případě je akceptovatelné alespoň zachování relace větší než. Tedy pokud $o(S_1, D) > o(S_2, D)$, pak i $o(S_1, D_z) > o(S_2, D_z)$.

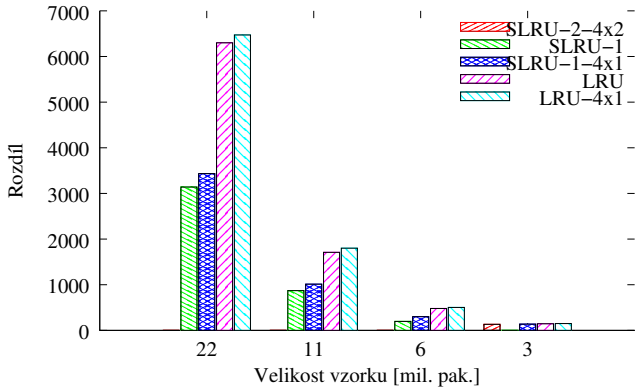


Obrázek 5.8: LRU, SLRU-1 a jejich modifikace.

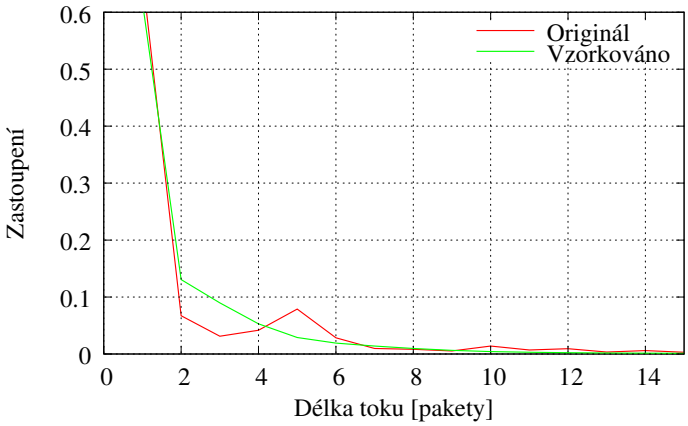
Za účelem určení dostatečné velikosti datové sady je proveden experiment, který má za úkol určit vhodnou velikost datové sady pro kvalitní ohodnocení správy cache z pohledu počtu odsunutých toků. Tento experiment využívá správy cache LRU a SLRU-1 (přípona -1 za SLRU značí $s = 1$, tedy pozici vkládání nových položek do řádku). Z těchto správ jsou vygenerovány modifikace (pozměněny definice jejich aktualizačních vektorů V_{LRU}, V_{SLRU}), které jsou ale původním vektorům velmi blízké, například se liší o jednu pozici místem vkládání nových stavů nebo o jednu pozici posunutým místem přesunu úspěšně vyhledaného stavu. Správa cache LRU je upravena tak, že stavy z posledních 4 pozic nejsou přesouvány na začátek řádku, ale až na druhou pozici v řádku. SLRU je upravena podobně tak, že stavy z posledních 4 pozic jsou přesouvány na místo vkládání nových stavů do řádku. Původní i nové modifikace správ cache jsou vyobrazeny na obrázku 5.8. Číselná přípona za zkratkou správy vyjadřuje, kolik pozic ve vektoru V na jakou hodnotu bylo změněno (například 4x2 značí změnu posledních čtyř pozic v řádku na hodnotu 2).

Délka řádku je nastavena na 32 stavů, velikost cache toků na 8192 stavů. Jako datový vzorek byla využita sada *Mawi-2010/04/14-14:00*. Tato datová sada obsahuje přibližně 44 mil. paketů a odpovídá 15 minutám provozu. Tato sada byla čtyřikrát po sobě zkrácena na polovinu směrem od jejího konce a vznikly tak sady nové o velikosti 22 mil., 11 mil., 6 mil. a 3 mil. paketů. Podobné správy cache jsou postupně simulovány na zkracující se datové sadě. Cílem je určit, zda je možné odlišit od sebe chování těchto správ, tj. sledovat rozdíl v počtu expirovaných stavů z cache. V grafu je vynesen rozdíl správ oproti nejlépe pracující správě na dané sadě. Obrázek 5.9 zobrazuje tyto rozdíly pro různé velikosti použité datové sady.

Z obrázku 5.9 je vidět, že pokud je vzorek dat příliš krátky, například 3 mil. paketů (kolem 70 s provozu), pak mezi podobnými správami není rozdíl. Dokonce nastala situace, kdy správa cache ohodnocená na delším vzorku jako horší, dosáhla nejlepšího ohodnocení na nejkratší sadě. S rostoucí délkou vzorku dat se rozdíl pomalu zvyšuje. Na základě tohoto experimentu lze usuzovat, že daný vzorek lze bezpečně zkrátit na čtvrtinu a s určitým rizikem i na osminu. Tento výsledek je platný pouze k danému nastavení cache toků a dané datové sadě. Jiná datová sada s jinou velikostí cache (či řádku) může dávat odlišné výsledky. Proto byly testovány i další nastavení na dalších datových sadách. Výsledky těchto experimentů ukazují, že pro simulaci datové sady *Snjc-2009/07/17-13:00* je nutné správu cache simulovat alespoň na dvouminutovém vzorku dat odpovídajícímu zhruba 50 milionům paketů. Na základě pozorování lze usoudit, že pro simulaci postačí využít datové sady, které obsahují alespoň 2 minuty provozu, pro velikost cache toků nastavenou tak, že



Obrázek 5.9: Rozdíl v počtu expirovaných stavů v závislosti na velikosti vzorku dat.

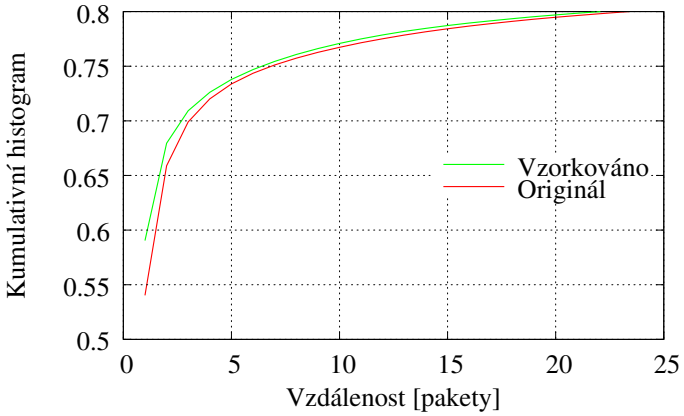


Obrázek 5.10: Příklad narušení charakteru rozložení délek toků v datové sadě vzorkováním provozu na úrovni paketů (datová sada *Mawi-2010/04/14-14:00*, $p_v = \frac{1}{2}$).

LRU je schopno dosáhnout méně než 10% výpadků.

Vzorek dat by místo zkrácení bylo možné navzorkovat, tj. vybrat pouze některé pakety ze vzorku. Vzorkování je možné provést s deterministickým výběrem paketů, kdy je pravidelně vybrán každý N-tý paket, nebo nedeterministicky, kdy paket je navzorkován s pravděpodobností $p_v = \frac{1}{N}$. Výsledný vzorek dat je N-krát menší než původní. Vyvstává ovšem otázka, zda navzorkovaná sada je reprezentativním obrazem původního sady. Graf 5.10 zobrazuje normalizovaný histogram délek toků původního a vzorkovaného provozu (hodnoty histogramu se normalizují celkovým počtem toků v původní, resp. navzorkované sadě). Přestože pravděpodobnost náhodného navzorkování je vysoká ($p_v = \frac{1}{2}$, v průměru každý druhý paket) dojde k odchýlení rozložení ve vzorkované sadě od původního rozložení toků. Zastoupení toků s jedním paketem zůstává zachováno. Na druhou stranu zastoupení toků se dvěma pakety se dvojnásobně zvyšuje. Tento jev je způsoben tím, že některé toky se třemi a více pakety se stávají vzorkováním toky se dvěma pakety. Zastoupení toků s pěti pakety se naopak snižuje na polovinu. Tato zkreslení mohou vést k tomu, že hledaná správa cache by byla optimalizována pro jiný druh provozu a na původním provozu by nepracovala tak dobře jako správa cache vyvíjena přímo pro danou síť.

Další možností je vzorkování na úrovni toků. To znamená vybrat všechny pakety navzorkovaných toků z celé datové sady a vytvořit z nich sadu novou. Tím bude zachován



Obrázek 5.11: Příklad porušení rozložení vzdáleností paketů v datové sadě vzorkováním provozu na úrovni toků ($p_v = \frac{1}{2}$).

histogram rozložení délek toků, přesněji bude zachován poměr zastoupení jednotlivých délek toků. Výběrem pouze některých toků, ale dojde k ovlivnění nepřímé interakce mezi souběžnými toky. Například, pokud se mezi pakety původního toku vyskytovaly pakety jiného toku, byl tento tok nepřímo tímto tokem ovlivněn. Snížením počtu toků tak může dojít ke zvýšení lokality vybraných toků a tím pádem i k ovlivnění vyvíjené správy cache. Navíc pokud bude sníženo množství toků v datové sadě, pak je nutné snížit i velikost cache toků. V opačném případě by mohlo dojít k tomu, že se stavy všech toků smístí do cache a nebude docházet k výpadkům. A tím pádem správa cache nebude vůbec zapotřebí.

Graf na obrázku 5.11 zobrazuje kumulativní distribuční funkci rozložení vzdáleností paketů pro původní a navzorkovanou sadu. Vzdálenost je měřena počtem paketů, které se nacházejí mezi dvěma následujícími pakety daného toku. Graf zobrazuje tuto vzdálenost *Mawi-2010/04/14-14:00*. Z grafu je patrné, že lokalita paketů v toku v originální sadě je horší než v sadě navzorkované, například v původní sadě byla pravděpodobnost, že následující dva pakety budou náležet stejnému toku pod 55%, kdežto v navzorkované sadě je tato pravděpodobnost kolem 60% při vzorkovacím poměru 1:2. Pro ohodnocení vyvíjené správy cache tedy nezbývá než zkrátit datovou sadu z pohledu doby trvání, tj. odstranit začátek či konec. Velikost sady je potřeba pro každou datovou sadu a konfiguraci cache toků určit na základě experimentů (lze doporučit využít alespoň 2 minuty provozu).

Vývoj správy cache může být rovněž urychlen, pokud je běh GA inicializován existujícími správami cache. V kapitole 4 bylo nastíněno několik správ cache. Některé z těchto správ lze dobře popsat navrženým formálním popisem. Jiné jsou naopak komplikovanější, neboť využívají dynamickou alokaci paměti (například pro uchovávání odkazů) anebo vyžadují komplexní paměťové struktury (například řadicí stromy). Pro následující experimenty nebude tato optimalizace využita. GA by v takovém případě mohl uváznout na lokálním optimu již na začátku běhu a selhal by při hledání inovativního řešení.

Kapitola 6

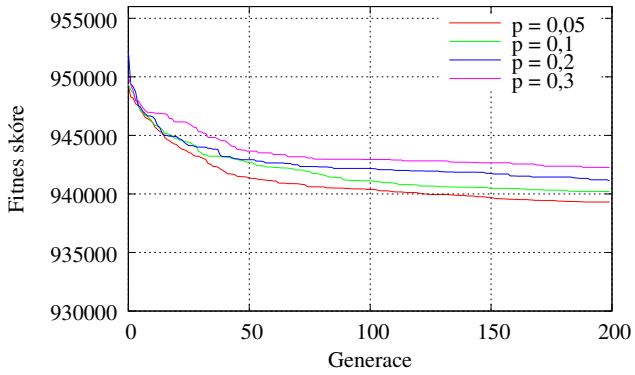
Experimenty

Tato kapitola je zaměřena na experimenty s genetickým algoritmem jako prostředkem pro vývoj správy cache toků. Genetický algoritmus má několik parametrů, které mají dopad na způsob prohledávání stavového prostoru a ovlivňují tak pravděpodobnost nalezení kvalitního řešení. Především se jedná o volbu a nastavení pravděpodobnosti operátorů mutace a křížení. Cílem je zjistit, jaký mají tato nastavení vliv na vývoj správy, a využít těchto poznatků pro rychlé nalezení kvalitních správ. Pro vývoj správy cache toků jsou ostatní parametry GA nastaveny následovně. Počáteční populace správ je vygenerována náhodně (uniformní rozložení). Prvky vektoru V nabývají hodnot v rozmezích dle definice daných jednotlivými optimalizacemi Opt_3 až Opt_6. Délka řádku je nastavena na $R = 32$. Velikost populace je nastavena na 6 jedinců vzhledem k časové náročnosti ohodnocení jedince. Relativně malá velikost populace tak umožňuje rychle ohodnotit danou populaci a přejít do dalších generací. Ohodnocení správ v populaci probíhá simulací sledování stavů toků na zkrácené datové sadě *Mawi-2010/04/14-14:00* s 6 miliony pakety. Výběr správ do nově vznikající populace probíhá turnajem, tj. náhodně jsou zvoleny dvě správy a správa s nižším ohodnocením je vybrána. Vybrané správy cache jsou s pravděpodobností p_{cross} zkříženy křížením a s pravděpodobností p_{mut} pozměněny mutací. Pro tvorbu nově vznikající populace byla zvolena generativní obměna celé populace s elitismem, tj. výběrem a zachováním nejlepšího jedince ze staré a nově vznikající populace. Pro každý z následujících experimentů je spuštěno deset instancí GA, aby bylo možné sledovat typický průběh vývoje. U jednotlivých experimentů je zaznamenán dopad využití jednotlivých optimalizací a příslušných operátorů na průběh vývoje správy.

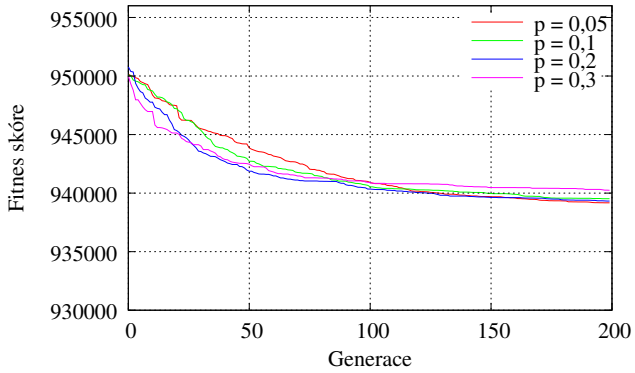
6.1 Vliv optimalizací a genetických operátorů

Snahou operátoru mutace je zvýšit diverzitu nově vznikající populace a nacházet tak nová řešení. Vhodná volba tohoto operátoru tak hraje klíčovou roli. V následujících experimentech jsou testovány vlastnosti různých operátorů mutace a vliv daného operátoru na kvalitu prohledávání stavového prostoru, tj. rychlost hledání a úspěšnost nalezené správy cache. V těchto experimentech je hledána správa cache, která se snaží o dosažení co nejmenšího počtu odstranění stavů toků z cache s ohodnocením jedinců dle výrazu 5.5. Pro následující experimenty s mutací je pravděpodobnost křížení $p_{cross}= 0$. U každého experimentu s mutačním operátorem je sledován průběh fitnes funkce v jednotlivých generacích. Především je sledován průběh ohodnocení nejlepšího jedince v průběhu evoluce.

Grafy na obrázcích 6.1, 6.2, 6.3, 6.4, 6.5 zobrazují průměr ohodnocení nejlepších



Obrázek 6.1: Průběh evoluce pro Opt_3 a O_{mut} s p_{mut} (průměr z deseti běhů).



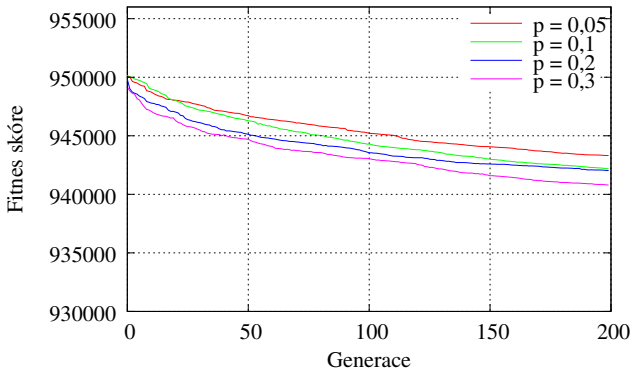
Obrázek 6.2: Průběh evoluce pro Opt_3 a O_{mut} s $p_{mut}(v)$.

jedinců v průběhu evoluce z jednotlivých běhů GA (pro každý experiment je provedeno 10 běhů).

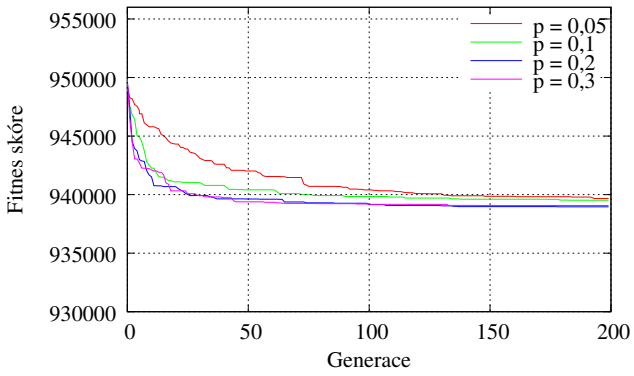
Graf na obrázku 6.1 zobrazuje průběhy pro operátor mutace O_{mut} pro čtyři nastavení pravděpodobnosti mutace p_{mut}. Z obrázku je patrné, že nejlepších výsledků dosahuje operátor mutace O_{mut} při $p_{mut}= 0,05$. Při detailní analýze jedinců v průběhu generací lze pozorovat, že vyšší pravděpodobnost mutace způsobí ve vektoru několik změn najednou. Přestože některé ze změn mohou vést k lepšímu řešení, výsledný vektor má horší hodnocení než původní kvůli zbývajícím změnám a to především v generacích v druhé půli běhu GA.

Další graf na obrázku 6.2 zachycuje experiment s mutačním operátorem O_{mut} s proměnnou pravděpodobností mutace $p_{mut}(v)$. Pravděpodobnost mutace roste lineárně se zvyšující se pozicí prvku vektoru V. Nejnižší hodnota pravděpodobnosti mutace je nastavena pro prvek v_2 na $p_{mut}(v_2) = 0,05$ a nejvyšší pro prvek v_{31} postupně na $p_{mut}(v_{31})= 0,05$, $p_{mut}(v_{31})= 0,1$, $p_{mut}(v_{31})= 0,2$, $p_{mut}(v_{31})= 0,3$. V grafu na obrázku 6.2 lze pozorovat, že průběhy ohodnocení i přes různá nastavení pravděpodobností mutace jsou podobné na rozdíl od ohodnocení pro předchozí pravděpodobnost $p_{mut}(v)$ na obrázku 6.1. Pravděpodobnost mutace $p_{mut}(v_{31}) = 0,3$ dovolila ze začátku běhu GA hledat rychleji lepší řešení a až v další části se projevil neblahý efekt vysoké pravděpodobnosti mutace.

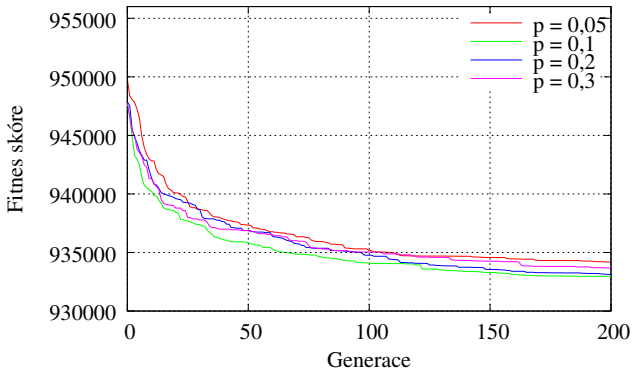
Graf na obrázku 6.3 zobrazuje experiment s operátorem mutace O_{mut-1} a pravděpodobností $p_{mut}(v)$. Při porovnání průběhu s předchozím obrázkem 6.1 lze pozorovat, že mutační operátor O_{mut-1} dosahuje horších výsledků, co se týká rychlosti průběhu prohledávání stavového prostoru i nalezeného nejlepšího řešení. Důvodem je pomalý vývoj vektoru, kdy jsou



Obrázek 6.3: Průběh evoluce pro Opt_3 a O_{mut-1} s $p_{mut}(v)$.



Obrázek 6.4: Průběh evoluce pro Opt_4 a mutaci O_{mut} s $p_{mut}(v)$.



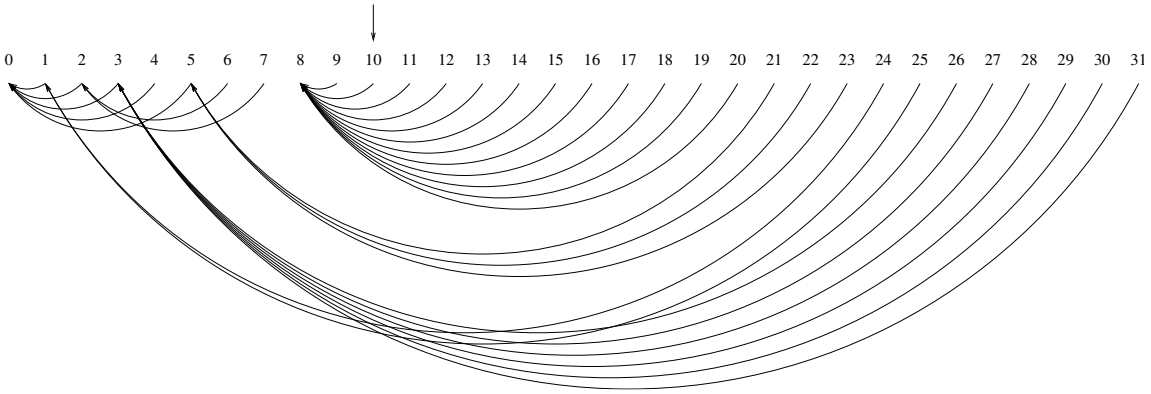
Obrázek 6.5: Průběh evoluce pro Opt_5 a mutaci O_{mut-w} s p_{mut}.

typicky generována velmi podobná řešení. Proto také vychází jako nejlepší zvolit vysokou míru mutace $p_{mut}(v)= 0,3$, která alespoň částečně zajistí generování inovativních jedinců. Pokud je změněn operátor tak, aby měl možnost zvýšit či snížit mutovaný prvek o číslo větší než jedna (na základě experimentů se nejlépe jeví zvolit číslo 3, operátor mutace je značen O_{mut-3}), pak se rychlost prohledávání stavového prostoru zvýší a GA rychleji dosáhne dobrých řešení. Zároveň při běhu této metody byla nalezena doposud nejlepší správa cache. Tato správa cache je zobrazena v příloze na obrázku 11.1.

Dále je proveden experiment s operátorem mutace O_{mut} s proměnnou pravděpodobností mutace $p_{mut}(v)$ a se zavedením optimalizace Opt_4. V grafu lze pozorovat, že již během prvních padesáti generací nalezne genetický algoritmus dobré řešení. To je způsobeno právě optimalizací Opt_4, která značně redukuje stavový prostor. Dále implementace mutačního operátoru s touto optimalizací dokáže v prvních generacích vyzkoušet velmi rozdílná, ale zato jednoduchá řešení, například vektory obsahující pouze několik odlišných hodnot prvků ve vektoru V. Ukazuje se, že souvislé části těchto vektorů se v dalších generacích stávají základem úspěšných správ cache. Většina běhů takto nastaveného GA vyvinula Segmented LRU správu cache, tedy vektor obsahující pouze prvky s hodnotou 0 a místem vkládání nových prvků na pozici 12. Při experimentu s mutačním operátorem O_{mut-3} v zakódování Opt_4 dosahuje GA obdobných výsledků až v pozdějších generacích. Znovu objevená správa cache SLRU podává velmi dobré výsledky jsou ovšem horší než výsledky správy cache při optimalizaci Opt_3 s O_{mut-3} a proměnnou pravděpodobností mutace $p_{mut}(v)$. Navíc při využití mutačního operátoru O_{mut-3} pro Opt_3 nebyla ani v jednom běhu v předešlých experimentech SLRU nalezena, přestože se jeví jako velmi slibná.

Na základě pozorování chování obou mutačních operátorů O_{mut-3} s optimalizací Opt_3 a O_{mut} s optimalizací Opt_4 a $p_{mut}(v)$ se jeví vhodné využít navrženou změnu reprezentace Opt_5. Tato reprezentace by měla umožnit generování jednoduchých souvislých struktur, podobných výsledkům GA s Opt_3 a operátorem O_{mut-3}. Zároveň odstraňuje omezení zavedené optimalizací Opt_4 a zachovává pouze Opt_3, tj. dovoluje přeskakování lepších pozic z pozic horších, které se jeví jako nezbytné pro dosažení lepších výsledků. Pro délku řádku $R = 32$ je zvolen počet úseků $Z = 8$.

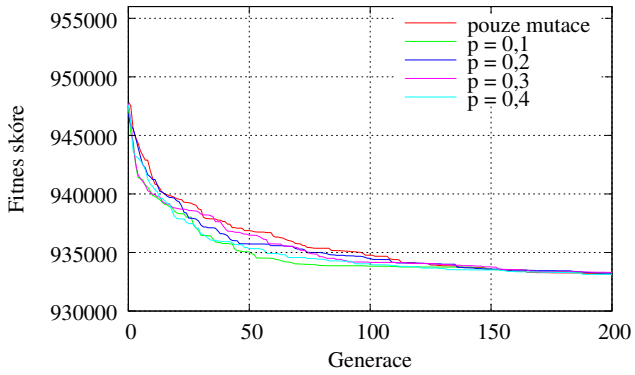
Z obrázku 6.5 je vidět, že vhodné zakódování umožňuje rychle najít slibné správy cache, které již po prvních 25 generacích překonávají řešení nalezená u jiných zakódování. Nejlepších výsledků dosahuje GA při nastavení pravděpodobnosti mutace na $p_{mut}= 0,1 \ldots 0,2$. Nejlepší nalezené řešení tímto operátorem je zobrazeno na obrázku 6.6 a pojmenováno jako GARP (*Genetic Algorithm Replacement Policy*).



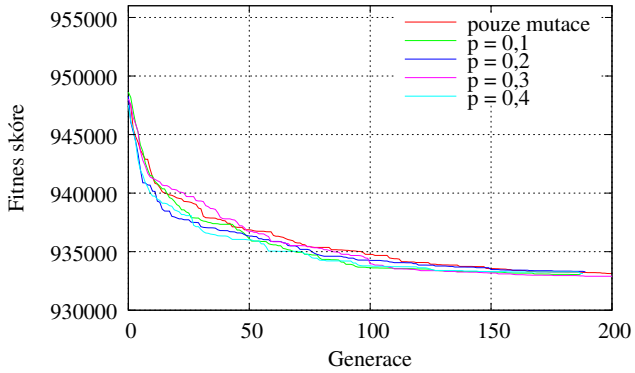
Obrázek 6.6: Grafické zobrazení správy cache nalezené pomocí GA. GARP = (10, (0, 0, 0, 0, 0, 0, 2, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 5, 5, 5, 1, 1, 3, 3, 3, 3, 3, 3)).

Znázorněná správa cache obsahuje šest souvislých úseků obsahujících stejnou hodnotu prvku, z toho jeden je delší než osm prvků (musí být složen z alespoň dvou úseků). První úsek přesouvá všechny prvky mezi nultou až pátou pozicí na samotný začátek seznamu. Další krátký úsek přesouvá z šesté a sedmé pozice záznamy na druhou pozici. Následuje dlouhý souvislý úsek, který přesouvá záznamy mezi pozicemi 8 až 20 na osmou pozici, zároveň jsou na pozici číslo 10 vkládány nové záznamy. Lze říci, že tento úsek tak tvoří SLRU. Následují úseky, které přesouvají aktualizované záznamy na pozice číslo 5, 1 a 3. Z vyvinuté správy cache můžeme usuzovat na model chování správy cache, který GA odvodil ze simulace chování síťových toků z pohledu příchodů paketů. Umístění pozice vkládání nových toků do zhruba 1/3 seznamu naznačuje, že existují toky, které je potřeba chránit před pakety nově příchozích toků, především těch obsahujících pouze jeden paket. Mezi takto chráněnými toky na pozicích 0 až 9 je nutné rozlišovat, tj. není vhodné aktualizované chráněné toky přesouvat na samotný začátek seznamu (na rozdíl od SLRU). Toky na pozicích 0 až 7 jsou pravděpodobně toky s dlouhými mezerami mezi pakety, neboť se do této části seznamu mohly dostat pouze z pozic 21 až 31. Pokud se vyskytují na prvních třech pozicích jsou jen velmi málo ohrožovány ostatními méně intenzivními toky. Pozice 8 a 9 je specifická, neboť chrání toky, které se pohybují v souvislém SLRU úseku (8 ... 20) před jednopaketovými toky, zároveň ale nedovolují aktualizovaným tokům z těchto pozic se přesunout do prvního úseku seznamu. Pokud je zavedeno dodatečně přemístění aktualizovaného toku z pozice 8 na pozici 7, pak dojde k narušení celého systému a správa cache dává znatelně horší výsledky. Toto chování si lze vysvětlit tak, že intezivní toky se přemístí do chráněných pozic a zaberou místo méně intenzivním, které využívají celou délku seznamu. Správa cache tedy využívá toho, že záznamy s dobrou lokalitou se v cache udrží, i když budou přeskakovány záznamy s horší lokalitou. Je vidět, že optimalizace Opt_4 by z tohoto pohledu zabránila takovému vývoji. Na pozicích 8 až 20 se vyskytují toky, které často obdrží paket a nepotřebují tak celou délku seznamu, aby odolaly jednopaketovým a málo intenzivním tokům. Z pozic 21 až 31 se dle významu přesouvají toky k začátku seznamu, tak aby mohly využít téměř celou délku seznamu než obdrží další paket. Z počtu souvislých úseků (celkem 6) lze usuzovat, že nastavení překódování aktualizačního vektoru V na 8 souvislých úseků je dostačující. Nevzniká tak potřeba počet úseků zvýšit.

Průběh GA s reprezentací pomocí vektoru Q při Opt_6 je obdobný průběhu Opt_5 v grafu 6.5, liší se pouze strmostí křivky, kdy Opt_6 prohledává stavový prostor pomaleji a dosáhne tak horších výsledků. Důvodem je prohledávání většího stavového prostoru rozšířeného



Obrázek 6.7: Průběh evoluce pro Opt_5 a $O_{cross-d}$ pro různé p_{cross}.



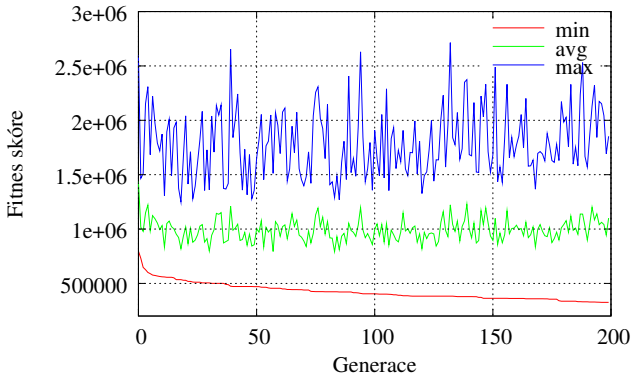
Obrázek 6.8: Průběh evoluce pro Opt_5 a $O_{cross-b}$ pro různé p_{cross}.

o inkrement q_j. Navíc se ve výsledku prvky q_j uplatní pouze zřídka a to na krátkých úsecích. Je možné tedy usoudit, že tyto prvky mají zanedbatelný vliv na samotnou kvalitu nalezené správy a spíše komplikují její rychlé nalezení. Z tohoto důvodu jsou experimenty s operátory křížení aplikovány pouze na zakódování Opt_5 v kombinaci s příslušnou mutací O_{mut-w} s pravděpodobností mutace $p_{mut}= 0,1$. Hodnota $p_{mut}= 0,1$ je zvolena proto, že s tímto nastavením bylo dosaženo nejlepších výsledků. Postupně jsou vyzkoušeny oba navržené operátory křížení.

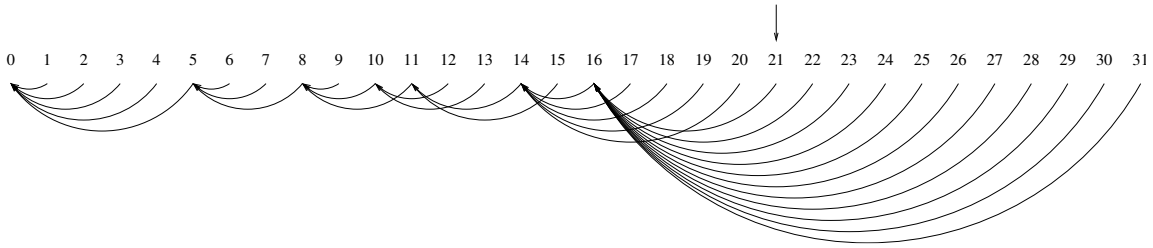
Na obrázcích 6.7, 6.8 jsou zobrazeny průběhy průměrů ohodnocení nejlepších jedinců v každé generaci pro různá nastavení pravděpodobnosti křížení p_{cross}. V grafech je záměrně zaznačen průběh GA pouze s mutací ($p_{cross}= 0$, $p_{mut}=0,1$). Porovnáním průběhů ostatních křivek s křivkou zobrazující průběh bez křížení, lze vidět, že křivky jsou svými průběhy velmi podobné. Můžeme tedy prohlásit, že ani jeden z navržených operátorů křížení nemá na průběh podstatný vliv. Mírné zlepšení průběhu lze pozorovat pro oba operátory $O_{cross-d}$ i $O_{cross-b}$ během prvních sto generací, následně se ale průběhy spojí a mají téměř shodný průběh.

Použitý operátor křížení není v této úloze operátorem, který by měl na výsledky hledání GA podstatný vliv. Tento jev byl pozorován i u mnoha jiných problémů, které byly řešeny pomocí GA. Důvod slabého dopadu křížení lze odůvodnit tím, že jednotlivé části vektoru jsou na sobě vzájemně závislé. Jejich rozdělení a výměna v rámci dalších vektorů nepřináší očekávaný pozitivní efekt šíření genetické informace v populaci.

Experimenty shrnuje tabulka 6.1. Tato tabulka zachycuje statistiku nad ohodnocením



Obrázek 6.9: Průběh evoluce pro Opt_5 a O_{mut-w} s p_{mut}.



Obrázek 6.10: Nalezená správa cache GARP-V1 = $(21, (0, 0, 0, 0, 0, 0, 5, 5, 5, 8, 8, 8, 10, 10, 11, 11, 14, 14, 14, 14, 14, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16))$.

(dle výrazu (5.5)) nejlepších správ paměti vyvinutých v jednotlivých běžích GA pro různá nastavení parametrů.

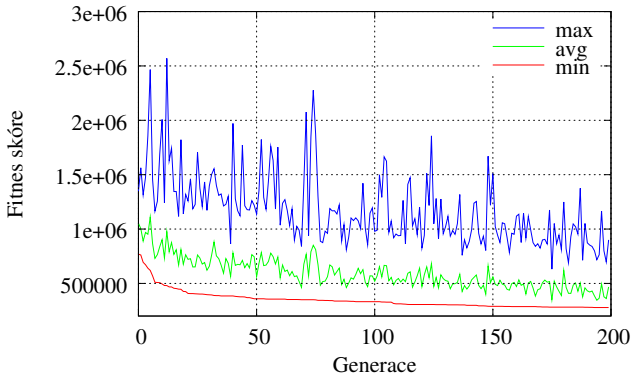
6.2 Vývoj správy cache velkých toků

Následující experimenty mají za cíl nalézt správu cache, která se dosáhne co nejmenšího počtu odstranění stavů velkých toků z cache. Jako fitness funkce je využíván výraz 5.7. Cílem experimentů je nalézt vhodné nastavení parametrů GA. Pro výchozí nastavení je zvolena konfigurace GA, která v předchozích experimentech v kapitole 6 nalezla nejlepší řešení správy. Je využíváno překódování aktualizačního vektoru V na vektor dvojic W délky $Z = 8$ splňující optimalizaci Opt_5. Jedinci jsou hodnoceni na základě součtu výpadků váhovaných podle kategorií dle výrazu 5.7. Výběr jedinců do další generace probíhá turnajem. Na vybrané jedince je aplikován operátor mutace O_{mut-w} na základě předchozích experimentů je pravděpodobnost mutace nastavena na $p_{mut-w} = 0{,}15$. Operátor křížení není využit. Náhrada populace probíhá generativní obměnou s elitismem. Běh GA je ukončen po 200 generacích. Pro každý experiment je spuštěno deset instancí GA. Průběh fitness funkce během evoluce (průměr minim, průměr maxim a průměr průměrů ze všech běhů) je zobrazen na obrázku 6.9.

Nejlépe ohodnocená správa vybraná z deseti běhů je zobrazena na obr. 6.10 a pojmenována GARP-V1. Vyvinutá správa velkých toků se významně liší od správy GARP na obr. 6.6, která byla vyvinuta za účelem celkového snížení výpadků stavů z cache. Markantní je jak rozdíl v pozici vkládání s nových stavů do řádku, tak i ve struktuře aktualizačního vektoru V. Zatímco u GARP je s situováno mezi první a druhou třetinou, u GARP-V1 je s mezi druhou a třetí třetinou. GARP-V1 vytváří velký prostor v řádku nalevo od místa

Tabulka 6.1: Statistiky ohodnocení správ snižujících celkový počet výpadků pro různá nastavení GA.

Nastavení					Fitnes dle dle výrazu (5.5)		
Opt	Mutace		Křížení		Min	Max	Průměr
Opt_3	O_{mut}	$p_{mut} = 0{,}05$	-	-	938326	941072	939308
Opt_3	O_{mut}	$p_{mut} = 0{,}1$	-	-	939244	942130	940208
Opt_3	O_{mut}	$p_{mut} = 0{,}2$	-	-	939968	941894	941108
Opt_3	O_{mut}	$p_{mut} = 0{,}3$	-	-	940942	943752	942264
Opt_3	O_{mut}	$p_{mut}(v) = 0{,}05$	-	-	938560	939946	939170
Opt_3	O_{mut}	$p_{mut}(v) = 0{,}1$	-	-	938142	940424	939524
Opt_3	O_{mut}	$p_{mut}(v) = 0{,}2$	-	-	938524	940582	939318
Opt_3	O_{mut}	$p_{mut}(v) = 0{,}3$	-	-	939162	941274	940240
Opt_3	O_{mut-1}	$p_{mut}(v) = 0{,}05$	-	-	941906	944320	943312
Opt_3	O_{mut-1}	$p_{mut}(v) = 0{,}1$	-	-	940460	946038	942184
Opt_3	O_{mut-1}	$p_{mut}(v) = 0{,}2$	-	-	939030	944266	942036
Opt_3	O_{mut-1}	$p_{mut}(v) = 0{,}3$	-	-	939604	941872	940788
Opt_4	O_{mut}	$p_{mut}(v) = 0{,}05$	-	-	938718	941188	939668
Opt_4	O_{mut}	$p_{mut}(v) = 0{,}1$	-	-	938718	942110	939524
Opt_4	O_{mut}	$p_{mut}(v) = 0{,}2$	-	-	938718	939474	938960
Opt_4	O_{mut}	$p_{mut}(v) = 0{,}3$	-	-	938718	939750	939072
Opt_4	O_{mut-1}	$p_{mut}(v) = 0{,}05$	-	-	943332	948484	945096
Opt_4	O_{mut-1}	$p_{mut}(v) = 0{,}1$	-	-	940820	943010	942022
Opt_4	O_{mut-1}	$p_{mut}(v) = 0{,}2$	-	-	939820	944636	942404
Opt_4	O_{mut-1}	$p_{mut}(v) = 0{,}3$	-	-	940652	945490	942932
Opt_5	O_{mut-w}	$p_{mut} = 0{,}05$	-	-	931736	936346	933444
Opt_5	O_{mut-w}	$p_{mut} = 0{,}1$	-	-	931524	933620	932540
Opt_5	O_{mut-w}	$p_{mut} = 0{,}2$	-	-	931460	936682	932644
Opt_5	O_{mut-w}	$p_{mut} = 0{,}3$	-	-	931970	934230	932952
Opt_6	O_{mut-w}	$p_{mut} = 0{,}05$	-	-	936551	942491	940521
Opt_6	O_{mut-w}	$p_{mut} = 0{,}1$	-	-	935340	939297	938030
Opt_6	O_{mut-w}	$p_{mut} = 0{,}2$	-	-	935285	940740	938949
Opt_6	O_{mut-w}	$p_{mut} = 0{,}3$	-	-	936792	941561	939384
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-b}$	$p_{cross} = 0{,}1$	932060	934614	933036
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-b}$	$p_{cross} = 0{,}2$	932056	934548	933272
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-b}$	$p_{cross} = 0{,}3$	932152	934978	932900
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-b}$	$p_{cross} = 0{,}4$	931952	935018	933136
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-d}$	$p_{cross} = 0{,}1$	931872	934470	932598
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-d}$	$p_{cross} = 0{,}2$	931768	933930	932706
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-d}$	$p_{cross} = 0{,}3$	931936	933278	932582
Opt_5	O_{mut-w}	$p_{mut}(v) = 0{,}15$	$O_{cross-d}$	$p_{cross} = 0{,}4$	932210	933748	932816



Obrázek 6.11: Průběh evoluce po překódování na Q vektor trojic $p_{mut} = 0,15$.

vkládání pro ochranu velkých toků před jednopaketovými toky. Jednopaketové toky jsou díky malé vzdálenosti pozice s od konce řádku rychle odstraněny. Složení vektoru V u GARP dovoluje přesouvat stavy z konce řádku až na začátek chráněné části. Naopak GARP-V1 je více konzervativní a stavy přesouvá po menších krocích směrem k začátku. Tento postupný přesun dovoluje další ochranu velkých toků. Pouze toky s větším množstvím paketů mohou soupeřit o pozici na začátku řádku. Stavy toků s malým množstvím paketů nemohou začátku řádku vůbec dosáhnout.

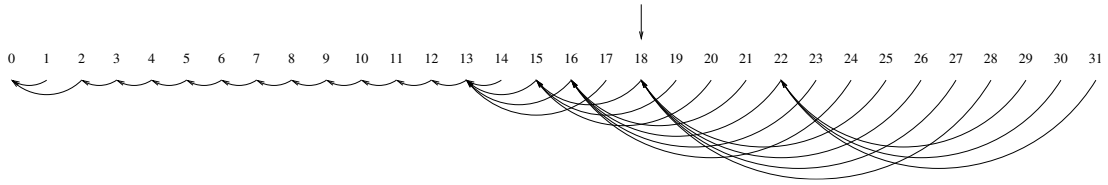
Průběh průměru maxim vykazuje mnoho lokálních extrémů. Zároveň je křivka maxima a průměru po celou dobu vývoje vzdálena od aktuálního minima. GA při hledání správy často generuje operátorem mutace řešení s velmi špatným ohodnocením. Při úpravě GA tak, aby docházelo pouze k částečné obměně jedinců mezi generacemi a bylo tak udržováno více slibných jedinců, průběh ohodnocení vykazuje méně extrémů, nicméně úspěšnost dosažení alespoň stejně dobrého řešení jako v základním nastavení klesá.

Ze složení vektoru V u GARP-V1 je patrné, že při reprezentaci pomocí vektoru dvojic W je všech $Z = 8$ souvislých úseků plně využito. Struktura nalezené GARP-V1 napovídá, že hodnoty prvků V by měly mít rostoucí tendenci. Z těchto pozorování lze odvodit potřebu navýšit počet dvojic vektoru W nebo využít zakódování do vektoru Q při využití Opt_6.

Pro délku řádku $R = 32$ je zachován počet posloupností $Z = 8$. Výsledek běhů experimentů s hledáním správy reprezentované vektorem Q je zobrazen na obr. 6.11. Nalezená správa je označena GARP-V2 a je zobrazena na obr. 6.12. GARP-V2 odráží potřebu správy cache jemně rozlišit stavy toků v chráněné části řádku. Toto rozlišení ve své podstatě realizuje počítání nedávných přístupů zakódované do pozice v řádku. Čím více přístupů, tím více je stav toku blíže začátku řádku. Experimenty shrnuje tabulka 6.2. Tato tabulka zachycuje statistiku nad ohodnocením (dle výrazu (5.7)) nejlepších správ paměti vyvinutých v jednotlivých bězích GA pro různá nastavení parametrů.

6.3 Porovnání s ostatními správami

Vyvinutá správa cache GARP (z obr. 6.6) je porovnána s ostatními správami. Pro porovnání výsledků jsou využity dvě charakteristiky M_1 a M_2. První z nich (M_1) je pravděpodobnost výpadků p_ν dle výrazu (5.6). M_2 zachycuje poměr počtu odstraněných toků z cache při simulaci vůči počtu toků v datové sadě. Tato charakteristika se využívá v síťové oblasti, neboť zachycuje průměrný počet výpadků na tok, tj. kolikrát byl odstraněn stav toku z cache před svým skutečným koncem.



Obrázek 6.12: Nalezená správa cache uchovávající velké toky nalezena při reprezentaci aktualizačniho vektoru pomocí vektoru Q. GARP-V2 = $(18, (0, 0, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 13, 13, 13, 15, 15, 15, 16, 16, 16, 16, 18, 18, 18, 18, 22, 22, 22))$.

Tabulka 6.2: Statistika nejlepších vyvinutých správ snižujících počet výpadků velkých toků pro jednotlivá nastavení GA.

Nastavení					Fitnes dle dle výrazu (5.5)		
Opt	Mutace		Křížení		Min	Max	Průměr
Opt_5	O_{mut-w}	$p_{mut} = 0{,}1$	-	-	212923	508091	321634
Opt_5	O_{mut-w}	$p_{mut} = 0{,}15$	-	-	208576	455678	306224
Opt_5	O_{mut-w}	$p_{mut} = 0{,}2$	-	-	214048	478339	318208
Opt_5	O_{mut-w}	$p_{mut} = 0{,}3$	-	-	238923	489245	347457
Opt_6	O_{mut-q}	$p_{mut} = 0{,}1$	-	-	211340	468273	294376
Opt_6	O_{mut-q}	$p_{mut} = 0{,}15$	-	-	201625	393301	278254
Opt_6	O_{mut-q}	$p_{mut} = 0{,}2$	-	-	222489	459327	305102
Opt_6	O_{mut-q}	$p_{mut} = 0{,}3$	-	-	221072	494318	348020

Pro porovnání správ cache jsou využity tři datové sady odlišné od sad popsané v úvodu a použitých pro vývoj pomocí GA. Konkrétně jsou využity datové sady, které byly nasbírány o 5 minut (*Snjc-13:05*, *Vut-15:05*) a 15 minut později (*Mawi-14:15*) než datové sady, na kterých GA hledal řešení. Tyto datové sady nebyly nijak upraveny nebo zkráceny a svými parametry jsou podobné datovým sadám *Mawi-2010/04/14-14:00*, *Snjc-2009/07/17-13:00*, *Snjc-2009/07/17-13:00*. Na základě experimentu je určena základní velikost cache toků tak, aby správa LRU dosahovala méně než 10% výpadků. Pro sadu *Mawi-14:15* a *Vut-15:05* postačí využít cache s 8192 stavy toků, pro *Snjc-13:05* je využita cache kapacitou 131072 stavů.

Jako referenční správa cache je uvedena FITF (Farthest in the Future), která pro danou velikost cache toků představuje optimální správu cache. FITF je správa cache založená na nápovědě, která pro každý tok obsahuje všechny časy příchodů paketů toku. V reálném nasazení není proto možné danou správu cache implementovat. FITF vybírá pro odsun z cache tok, jehož následující paket v budoucnosti je nejdále v porovnání s ostatními následujícími pakety toků. Velikost cache omezuje počet stavů, které mohou být v cache uloženy. Pokud počet současně aktivních toků je větší než velikost cache, pak ani optimální správa cache nedokáže udržet všechny stavy toků bez výpadků. Nicméně poskytuje informaci o úspěšnosti, které lze dosáhnout s danou velikostí cache. Tomuto limitu se pak snaží ostatní správy přiblížit.

Pro každou datovou sadu je nalezena odlišná správa (nalezené správy jsou uvedeny v příloze v tabulce 11.2). Pro porovnání výsledků s GARP jsou vybrány správy cache, které jsou běžně implementovány v současných zařízeních. Především se jedná o populární LRU. Dále jsou vybrány správy cache, které lze považovat za špičku v dané kategorii správ

Tabulka 6.3: Porovnání úspěšnosti správ cache.

	Mawi-14:15		*Snjc-13:05*		*Vut-15:05*	
Správa	M_1	M_2	M_1	M_2	M_1	M_2
EXP1	18,8	257,1	17,7	298,3	19,9	600,5
LFU*-aging	8,3	169,1	5,8	163,8	2,5	162,6
LIRS	7,8	164,2	6,8	174,3	3,1	177,1
LRU-3	6,1	150,8	4,5	148,6	2,4	159,2
LRU-2	6,1	150,8	4,5	148,6	2,4	159,2
LRU	6,1	150,8	4,5	148,6	2,4	159,2
SLRU	6	149,1	4,3	146,4	2,3	157,7
GARP	5,3	144,8	4,1	144,1	2	151,6
FITF	1,9	115,6	0,9	109,3	0,9	122,7

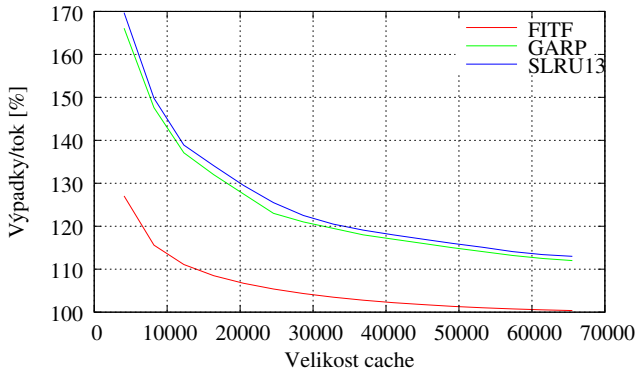
cache, přestože výpočetní složitost je vysoká a jejich implementace v hardware by byla velmi komplikovaná. Kategorie založena na sledování nedávných přístupů je zastoupena správami LRU-2, LRU-3, LIRS. Kategorii správ založených na frekvenci přístupů zastupuje EXP1 a LFU*-aging. Tabulka 6.3 uvádí výsledky pro použité datové sady a správy cache. Pro každou datovou sadu jsou ve sloupcích uvedeny pravděpodobnosti výpadku stavů toků z cache a následně poměry počtu odsunutých stavů toků vůči skutečnému počtu unikátních toků vyjádřený procenty.

Z výsledků v tabulce je patrné, že nejhůře skončila správa EXP1. Tento výsledek lze očekávat, neboť správy cache spravované dle frekvence přístupů jsou vhodné pro datové sady s dlouho existujícími objekty (například webové stránky v tzv. web cache). EXP1 počítá na základě předchozích přístupů odhad následujícího přístupu μ_i ke stavu toku. Tento odhad je počítán na základě poslední délky intervalu t_{i-1} a předchozího odhadu μ_{i-1} jako $\mu_i = \alpha t_{i-1} + (1-\alpha)\mu_{i-1}$, kde α je parametr váhy předchozího odhadu a na základě rozboru autorů je vhodné tento parametr nastavit na $\alpha = 0,1$. Na základě experimentů s nastavením parametru α bylo zjištěno, že pro síťový provoz je vhodnější nastavit $\alpha = 0,9$. Přesto EXP1 do značné míry selhala jako správa cache vhodná pro správu cache toků.

Správa cache LFU*-aging dopadla mnohem lépe, přestože náleží rovněž do skupiny správ cache zaměřených na sledování frekvence. Tato správa ale obsahuje mechanismy, které ji umožňují přizpůsobit se aktuálnímu provozu. LFU*-aging zabraňuje přílišné prioritizaci frekventovaně přistupovaných stavů omezením maximální hodnoty čítače přístupů dle parametru M_{ref}. Dále odstraňuje pouze stavy s maximálně jedním přístupem. Pokud takový stav není k dispozici, pak nevloží vůbec nový stav toku do cache. A třetí vlasností je stárnutí uložených stavů pomocí zmenšení hodnoty čítačů na polovinu po každé po A_{max} přístupech do cache. V provedených experimentech bylo nastaveno $M_{ref} = 10$ a $A_{max} = 10000$.

Následně se umístila správa cache LIRS pro datovou sadu *Mawi-14:15* v datové sadě *Snjc-13:05* je pořadí LIR a LFU*-aging prohozeno. Parametr α určující poměr uchovávaných LIR a HIR položek u této správy cache byl nastaven podle doporučení autorů LIR na $\alpha = 0,9$, což se ukázalo jako vhodné nastavení pro síťový provoz. Nicméně LIRS nedokázala překonat výsledky běžné LRU i přes to, že obsahuje poměrně složité mechanismy pro určení, které stavy v cache uchovat a které odstranit.

Správa cache LRU-K byla testována ve třech variantách. Varianty využívající historii tří resp. dvou přístupů (LRU-3 a LRU-2) nedosáhly očekávaných zlepšení. Jejich výsledky byly naprosto totožné s běžnou správou LRU (LRU-1). Tento výsledek je poměrně zajímavý



Obrázek 6.13: Procentuální poměr odsunutých toků vůči skutečnému počtu toků (M_2 pro vybrané správy cache na datové sadě *Mawi-14:15*.

Tabulka 6.4: Rozdíl mezi vektory pro různé velikosti pamětí (*Mawi-2010/04/14-14:00*).

	4K - 8K	8K - 12K	12K - 16K
rozdíl vektorů V	3,03	1,25	0,65
rozdíl pozice s	3	3	2

a naznačuje, že s se správou LRU není možné využít historii předchozích přístupů až na přístup poslední pro předpověď přístupu následujícího.

Nejlépe z reálně implementovatelných správ se umístila správa GARP, za ní správa SLRU. GARP tedy překonala ostatní doposud navržené heuristiky. Rozdíl výsledků GARP a SLRU je v řádu desetin procent, nicméně větší než rozdíl SLRU, LRU a LRU-N.

Při porovnání výsledků GARP a FITF je patrné, že existuje prostor pro další optimalizace správy GARP směrem k FITF. Na základě poměrně malých rozdílů v úspěšnosti reálných správ paměti lze předpokládat, že další optimalizace nemůže být založena pouze na sledování historie přístupů. Znalost historie přístupů poskytuje dobrý základ pro predikci budoucích přístupů, nicméně nemůže zachytit specifické události síťového provozu jako je například náhlé přerušení spojení. Dalšímu rozšiřování správy cache se věnuje kapitola 8.

Úspěšnost nalezení stavu toku v cache je závislá nejen na správě cache, ale především i na velikosti cache. Obrázek 6.13 zobrazuje procentuální poměr počtu odsunutých stavů toků vůči skutečnému počtu toků pro různé velikosti cache toků pro tři nejlépe pracující správy cache.

Pro každou velikost cache se bude lišit i nalezená správa GARP. Příklady výsledných správ GARP pro pro datovou sadu *Mawi-2010/04/14-14:00* a rozdílné velikosti pamětí jsou uvedeny v příloze tabulce 11.3. Tabulka 6.4 vyjadřuje jejich vzájemnou odlišnost jako průměr absolutních hodnot rozdílů odpovídajících si prvků aktualizačních vektorů V a dále rozdíl vstupní pozice.

Nejmarkantnější rozdíl je mezi GARP správami vyvinutými pro cache s kapacitou 4096 a 8192 stavů. Příčinou tohoto rozdílu je nepoměr mezi velikostí cache a počtem současně aktivních toků. Jak je vidět z obrázku 6.13, ideální správa cache je schopna dosáhnout téměř 0% pravděpodobnosti výpadků až při velikosti cache 64K stavů. To je 15-krát více než nejmenší testovaná velikost 4096 stavů. GARP správa pro tak malou cache se proto musí těmto podmínkám přizpůsobit. Pokud je tato správa cache použita na větší paměť, například pro paměť 16K, pak podává horší výsledky (cca o 0,3% horší úspěšnost vyhledání)

než GARP správa vyvinuta pro tuto velikost.

Pro porovnání vlasností správ velkých toků jsou zvoleny dvě charakteristiky (M_3 a M_4), které odráží cíle sledování velkých toků. M_3 zachycuje procento toků v jednotlivých kategoriích, které zaznamenaly alespoň jeden výpadek (výraz 5.8), který není způsoben prvním paketem toku. M_3 odráží cíl udržet stavy velkých toků v cache bez výpadku tak, aby byla odstraněna potřeba záložní paměti. M_4 zachycuje snížení počtu výpadků u velkých toků (výraz 5.7). Výsledek je reprezentovaný jako procentuální poměr výpadků v dané kategorii vůči počtu toků v této kategorii. M_4 tak doplňuje první charakteristiku informací o průměrném počtu výpadků na tok dané kategorie.

Pro porovnání jsou vybrány často používané správy anebo správy, u kterých se předpokládá zaměření na velké toky. LRU je nejčastější správou implementovanou v komerčních zařízeních [34]. Každá cache, která odstraňuje položky na základě překročení doby neaktivního intervalu, musí nutně implementovat LRU. Správy LRU a SLRU pracují při běžném síťovém provozu velmi blízko dosažitelného optima, jak lze vidět z výsledků v kapitole 6.3. Mezi další zkoumané správy je zařazena LRU-2, která by na datových sadách se špatnou lokalitou měla podávat lepší nebo stejné výsledky jako LRU. Správa paměti LIRS byla navržena pro snížení počtu výpadků dat z cache na datových sadách se specifickými vzory přístupů. Ze skupiny správ založených na sledování počtu přístupů se nejlépe osvědčila LFU*-aging. S^3-LRU-7 je správa paměti navržena pro sledování velkých toků v cache [64].

Tabulky 6.5, 6.6, 6.7 zobrazují výsledky správ na třech různých vzorcích provozu. Na prvních dvou sadách je nejhorší správou LFU*-aging. Na třetí sadě ale naopak tato správa podává dobré výsledky. Příčina tohoto rozdílného chování spočívá ve specifickém složení sady *Vut-15:05*. Tato sada obsahuje několik velmi velkých toků, které přenáší většinu provozu. V porovnání s jinými sadami je trvání těchto toků delší. Správa LFU*-aging byla s ohledem na tyto vlasnosti navržena. Správa LIRS podává lepší výsledky oproti LFU*-aging na prvních dvou sadách. Na třetí sadě je ale naopak horší, neboť tato správa není úzce zaměřena na sledování jen největších toků. Podobně správy LRU a LRU-2 nejsou schopny uchovat stavy velkých toků bez přerušení v cache a způsobují velké množství výpadků stavů velkých toků. Tyto správy vkládají všechny nové stavy na začátek řádku, stavy tak vydrží déle v cache a mají vyšší šanci na přijetí dalšího paketu. Nicméně odsouvají již stávající stavy a způsobují tak výpadky stavů velkých toků. Proto je u správy SLRU vhodné posunout pozici s vkládání dále směrem ke konci řádku (jako nejlepší pozice s ohledem na výraz 5.2 je experimenty zjištěna pozice 21). Chráněný prostor je větší a SLRU-21 generuje méně výpadků stavů velkých toků než LRU. Velmi dobře se umístila S^3-LRU-7, především z pohledu kategorie L_1. U dalších kategorií toků dosahuje S^3-LRU-7 horších výsledků. Důvodem je úzké zaměření na stavy velkých toků, které postupně přesouvá do chráněné části řádku. Pozice vkládání nových stavů je u S^3-LRU-7 nastavena na základě experimentů s ohledem na výraz 5.2. Nejlépe z online správ se umístila GARP-V2 v obou charakteristikách napříč všemi datovými sadami. FITF-V reprezentuje ideální offline správu cache. Tato správa během prvního průchodu datovou sadou zaznamená časy příchodů paketů. Na konci prvního průchodu vyhodnotí příslušnost toků do jednotlivých kategorií. Během druhého průchodu spravuje cache následovně. Pokud je potřeba vytvořit místo v cache pro stav nového toku, pak FITF-V hledá a odstraní stav náležící postupně kategorii L_4, L_3, L_2, L_1. V případě možnosti výběru z dané kategorie odstraní stav, který nebude nejdéle přistoupen. Za předpokladu, že by počet velkých toků byl menší než kapacita paměti, pak je FITF-V schopna uchovat stavy velkých toků bez jediného výpadku. Tento stav nastal u sady *Snjc-13:05*. Ostatní datové obsahovaly více souběžných toků kategorie L_1, L_2, L_3.

Tabulka 6.5: Porovnání úspěšnosti správ cache velkých toků *Mawi-14:15*.

	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
LFU*-aging	29%	37%	21%	79%	91%	85%
LIRS	21%	33%	27%	50%	88%	71%
LRU-2	19%	30%	10%	79%	95%	48%
LRU	19%	30%	10%	79%	95%	48%
SLRU-21	14%	17%	8%	33%	47%	22%
S^3-LRU-7	5%	14%	21%	26%	68%	55%
GARP-V2	4%	8%	12%	9%	21%	34%
FITF-V	0%	1%	6%	0%	2%	36%

Tabulka 6.6: Porovnání úspěšnosti správ cache velkých toků *Snjc-13:05*.

	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
LFU*-aging	0%	29%	31%	0%	93%	95%
LIRS	0%	22%	36%	0%	51%	86%
LRU-2	0%	24%	22%	0%	76%	62%
LRU	0%	24%	22%	0%	76%	62%
SLRU-21	0%	19%	14%	0%	54%	37%
S^3-LRU-7	0%	15%	18%	0%	40%	47%
GARP-V2	0%	7%	9%	0%	18%	17%
FITF-V	0%	0%	0%	0%	0%	0%

Obrázek 6.14 ukazuje chování GARP-V2 pro různé velikosti cache na datové sadě *Mawi-14:15*. Se zvyšující se velikostí cache poměr výpadků klesá exponenciálně pro všechny kategorie. Od velikosti cache vyšší než 4096 dochází již jen k pozvolnému snižování výpadků. Detailní analýza chování GARP-V2 odhaluje, že tyto výpadky nastávají v převážné většině případů (přibližně 70% dle datové sady) na začátku toku, kdy teprve začíná intenzivní přenos dat. Navíc stav toku se v této fázi velmi pravděpodobně nachází napravo od bodu vložení a pravděpodobnost jeho odstranění je tak vyšší. Pokud velký tok zaznamená výpadek, pak v 90% případů se jedná o jediný výpadek během celého toku. Posun bodu vložení směrem k začátku by zvýšil šanci stavu velkého toku obdržet další paket, který by posunul stav do chráněné části. Nicméně by se tím chráněná část stala menší a toky s jedním paketem by zabíraly v cache místo. Nalezená pozice bodu vložení s je tak kompromisem odpovídající charakteristikám datové sady.

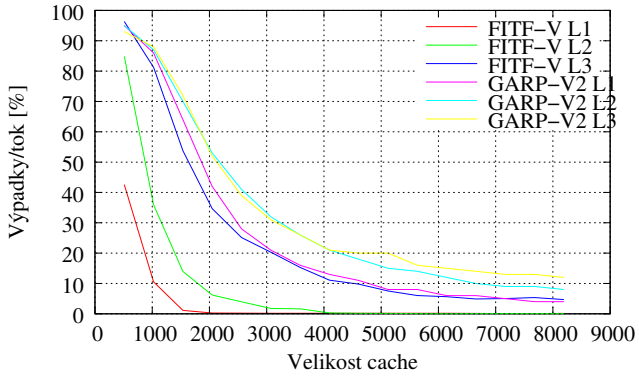
6.4 Adaptace správy cache

Složení síťového provozu se neustále vyvíjí. Tento vývoj je řízen nejen rozšiřováním Internetu mezi více uživatelů a zvyšováním rychlostí linek, ale především aplikacemi a službami, které na Internetu uživatelé využívají. Mezi významné změny v historii Internetu patří například rozšíření p2p aplikací pro sdílení obsahu mezi uživateli.

Změna ve složení aplikací, které jsou na Internetu využívány, způsobí i změnu vlastností

Tabulka 6.7: Porovnání úspěšnosti správ cache velkých toků *Vut-15:05*.

	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
LFU*-aging	9%	19%	25%	22%	48%	69%
LIRS	17%	21%	27%	39%	53%	74%
LRU-2	29%	42%	37%	223%	142%	112%
LRU	31%	45%	37%	220%	150%	112%
SLRU-21	21%	31%	31%	59%	90%	89%
S^3-LRU-7	18%	32%	32%	58%	89%	97%
GARP-V2	7%	11%	20%	5%	7%	19%
FITF-V	2%	3%	8%	3%	6%	19%



Obrázek 6.14: Procentuální poměr výpadků vůči počtu toků pro různé velikosti cache.

síťového provozu. Tato změna může mít za následek nižší úspěšnost správy cache. Proto je důležité, aby se správa cache dokázala přizpůsobit aktuálnímu složení síťového provozu. GARP tuto možnost poskytuje. Je možné opětovně spouštět proces hledání nové správy, která bude dosahovat lepších výsledků než stávající správa. Zůstává otázkou, jak často je nutné správu cache přizpůsobit a zda lze pozorovat tzv. efekt stárnutí.

Pro ověření efektu stárnutí správy cache je proveden následující experiment. GA nalezne správy pamětí postupně na několika datových sadách provozu stejné linky ale časově vzdálených od sebe několik dní až roků. Každá správa je tak optimalizována pro danou sadu. Všechny správy jsou následně ohodnoceny na vlastní datové sadě, která sloužila pro nalezení správy cache, a na ostatních (cizích) datových sadách, pro které nebyly správy optimalizovány. Na základě výsledků správ cache na cizích sadách je možné usuzovat na rychlost stárnutí správy cache.

Tabulka 6.8 zobrazuje výsledky tohoto experimentu. Horizontálně jsou vyneseny výsledky správ pamětí pro danou datovou sadu (datum datové sady je uvedeno na začátku každého řádku). Vertikálně pak můžeme získat výsledky pro danou správu cache přes všechny datové sady. Číslo správy cache uvedené horizontálně v záhlaví tabulky odpovídá číslu datové sady, pro kterou byla daná správa nalezena. Výsledky značí pořadí dané správy cache z pohledu úspěšnosti vyhledání záznamů v cache. Pro datovou sadu můžeme v daném řádku vyhledat nejlepší správu cache označenou číslem 0 a nejhorší správu cache označenou číslem 8.

Je důležité poznamenat, že absolutní hodnoty úspěšnosti vyhledání prvku v cache se pro

Tabulka 6.8: Úspěšnost správy pamětí ohodnocených na cizích datových sadách (datová sada zachycena vždy ve stejný čas 14:00 z linky Mawi).

Číslo	Datum	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
1.	21.3.2008	0	7	4	3	2	1	6	9	5	8
2.	28.3.2008	4	0	5	2	3	1	6	9	7	8
3.	4.4.2008	5	3	1	4	2	0	6	9	7	8
4.	11.4.2008	4	1	5	0	3	2	6	9	7	8
5.	12.5.2008	1	4	7	2	0	3	6	9	5	8
6.	11.6.2008	5	2	1	4	3	0	6	9	8	7
7.	11.12.2008	7	6	4	8	3	5	1	9	2	0
8.	11.6.2009	8	7	4	9	6	5	2	0	3	1
9.	11.12.2009	6	8	5	7	3	4	1	9	0	2
10.	11.12.2010	6	8	3	7	5	4	1	9	2	0

nejhorší a nejlepší správu GARP lišily ve většině případů o 0,2%. Vyjímku tvoří výsledky na sadě z 11.6.2009, kde rozdíl nejlepší a nejhorší správy cache je 1,1%. Všechny správy nalezené pomocí GA vždy překonaly ostatní správy paměti.

Při pohledu na matici výsledků v předchozí tabulce si lze všimnout několika zajímavých jevů. Jedním z nich je, že na hlavní diagonále (vyznačena červeně) se nachází buď 0 či 1. To znamená, že správa cache vyvinutá na dané datové sadě je pro tuto datovou sadu nejlepší či druhá nejlepší v porovnání se správami vyvinutých na ostatních datových sadách. Na cizích datových sadách je úspěšnost správy cache špatně předvídatelná. Velmi zajímavý výsledek je možné vidět v sloupci číslo 8, ve kterém daná správa cache je naprosto nejhorší na cizích datových sadách, ale na vlastní datové sadě je nejlepší. Z tohoto jevu je možné usuzovat, že daná datová sada obsahovala velmi specifické vzory chování (například útok na odepření služby pomocí zahlcení zdrojů). Výsledná správa cache se tak výrazně liší od ostatních. Po vykreslení této správy do obrázku 11.2 v příloze je možné pozorovat značné rozdíly oproti správě cache vyobrazené na obr. 6.6. Průměrně se prvky vektoru V liší o 4,3 pozice.

Dále zaměřme pozornost na stárnutí určité správy cache v průběhu času. Pokud by docházelo k pozvolnému stárnutí, pak by se ve sloupcích objevovaly postupně se zvyšující hodnoty pod a nad hlavní diagonálou, čím dále od diagonály tím horší umístění. Z výsledků je ale patrné, že spíše než ke stárnutí dochází k tomu, že správa cache je přizpůsobena konkrétní datové sadě, ale ke stárnutí jako takovému dochází velmi pomalu, pokud vůbec. Pro hledání správy je vhodné vybrat takovou datovou sadu, která reprezentuje složení síťového provozu po většinu doby. Toho lze docílit vhodným výběrem datové sady na základě fundamentální znalosti chování síťového provozu v kombinaci s technickou analýzou.

Výběr datové sady může být proveden na základě spuštění několika instancí GA nad různými datovými sadami, které svou velikostí postačují pro ohodnocení kandidátní správy cache. Poté co instance GA naleznou správy cache pro svou vlastní sadu, nalezené správy cache jsou ohodnoceny na cizích datových sadách. Takové ohodnocení odpovídá uvedené tabulce 6.8. Vzájemné ohodnocení je časově méně náročné (odpovídá ohodnocení několika populacím) než samotný běh GA. Na základě této tabulky lze vybrat správu cache, která v průměru podává nejlépší výsledky přes všechny datové sady. V případě tabulky 6.8 je to správa cache nalezená na datové sadě číslo 7. Tato správa se průměrné umísťuje na 2. až 3. místě přes všechny datové sady. Je tak velmi pravděpodobné, že datová sada číslo 7 svým složením nejlépe zachycuje vzory chování provozu na dané lince a může dlouhodobě sloužit

Tabulka 6.9: Rozdíl mezi vektory pro různé datové sady.

	Mawi-Vut	Vut - Snjc	Snjc - Mawi
rozdíl vektorů V	1	0	1
rozdíl pozice s	0	1	1

pro správu cache v zařízení nasazeném na dané síti.

Při porovnání vyvinutých správ velkých toků na rozdílných datových sadách se ukazuje, že architektura správ (pozice vkládání a aktualizační vektor) se liší velmi málo. Značnou podobnost správ lze připsat jejich zaměření na velké toky. Přestože každá sada má odlišné rozložení toků v kategoriích, všechny vykazují *heavy-tail* rozložení počtu paketů v tocích. Správy cache velkých toků vyvinutých na různých datových sadách jsou zobrazeny v příloze v tabulce 11.5. Tabulka 6.9 vyjadřuje jejich vzájemnou odlišnost jako průměr absolutních hodnot rozdílů odpovídajících si prvků aktualizačních vektorů V a dále rozdíl vstupní pozice.

Vzhledem k malým rozdílům mezi správami různých datových sad je možné předpokládat, že rozdíl mezi správami paměti vyvinutých na vzdálených datových sadách bude rovněž malý. Tento předpoklad je experimentálně potvrzen vývojem správy paměti na několika vzorcích provozu jedné linky, které jsou od sebe vzdáleny různě dlouhé intervaly v čase. Porovnání vektorů je prezentováno v příloze v tabulce 11.5.

Vývoj správy přímo na reálném provozu na zařízení, které je nasazeno a využíváno na síti, ztěžuje několik skutečností. První z nich je výkon a velikost dostupné cache v takovém zařízení. Plná podpora GA běžícího na reálných datech by vyžadovala, aby zařízení dovolilo GA vyvíjet správu cache jako nezávislý proces běžící na pozadí a neovlivňující jeho běžnou činnost. K tomu by bylo zapotřebí dostatek výkonu ale také velké množství běžně nevyužívané cache, ve které by mohla být kandidátní správa cache ohodnocena. Tyto parametry ale produkční zařízení neposkytují, naopak nároky na paměť často převyšují dostupnou kapacitu. Další skutečností jsou výkyvy množství dat a jejich vlasností během běhu GA. Pokud pro ohodnocení jednoho jedince jsou potřeba alespoň čtyři minuty provozu, pak pro běh s 200 generacemi a celkově 1000 ohodnocenými jedinci dostáváme čas běhu 66 hodin. Vzpomeňme si ale na periodický nárůst a pokles množství toků a provozu během dne (obr. 3.1). Správa cache, která bude ohodnocena během dopoledne, kdy množství provozu dosahuje nejvyšších hodnot, bude mít nesrovnatelné výsledky se správou ohodnocenou na provozu o půlnoci. Navíc experimenty s během GA na střídajících se datových sadách (tři datové sady s rozdílnými vlastnostmi) ukázaly, že GA není schopen bez dalších úprav správně prohledávat stavový prostor řešení. Pokud by tedy bylo nutné správu cache vyvíjet, pak by běh GA měl využívat pouze jeden vzorek dat. Volba tohoto vzorku je ale kritická pro vývoj správy cache, která bude optimalizována pro více typů chování provozu na síti. Z tohoto pohledu se jeví jako vhodné sledovat pouze úspěšnost správy na nasazeném zařízení, a pokud klesne pod určitou mez, pak spustit vývoj nové správy mimo zařízení na několika vzorcích provozu získaných z dané sítě.

Kapitola 7

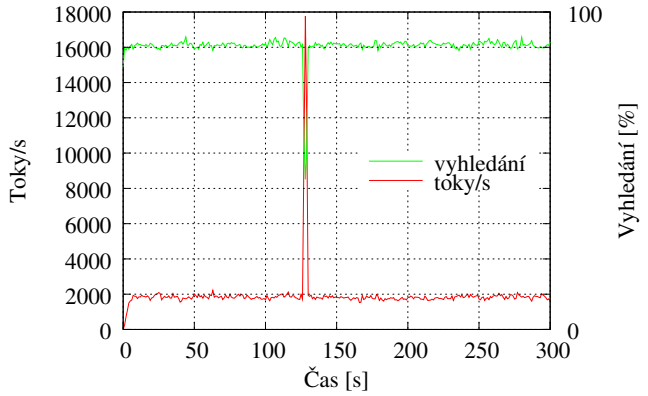
Správa paměti ve světle útoků

Cache toků je kritickým místem při stavovém zpracování síťového provozu. Pokud se většina stavů toků nebude nacházet v cache, pak zpracování paketu musí čekat na získání stavu ze záložní paměti nebo výpočetního prvku. Úspěšnost nalezení stavu v cache toků má podstatný vliv na výkonnost celého systému. Při výběru správy cache je proto důležité zvážit možnost výskytu různých typů síťového provozu.

Správně dimenzovaná cache toků se zvolenou správou dosahuje za běžného provozu vysoké úspěšnosti vyhledání stavů v cache. Výskyt náhlého a velkého nárůstu toků, který se v legitimním provozu nevyskytuje, může způsobit selhání správy a pokles úspěšnosti vyhledání stavu v cache. Náhlé změny ve složení provozu bývají vyvolány aktivitou spojenou nelegálním chováním na síti. Typicky se jedná o distribuované útoky na odepření služby (*DDoS - Distributed Denial of Service*) nebo skenování a zjišťování informací o síti (*network scan*). Cílem DoS útoků je vyčerpat paměťové zdroje zařízení zpracovávající tyto toky. S příchodem nového toku je vytvořen stav v paměti, který je uchován, dokud nevyprší přednastavený interval. Pokud přichází dostatek nových toků, dojde k vyčerpání dostupné paměti v zařízení. Toto chování se projevuje výskytem mnoha toků s malým množstvím paketů. Malé množství paketů v toku umožňuje zdroji anomálie vygenerovat velké množství toků i při omezené přenosové kapacitě linky. Nejznámnějším typem útoku je záplava TCP SYN pakety, tj. pakety pokoušející se o navázání spojení s TCP serverem. Server založí TCB a odpoví žadateli o spojení. V případě útoku žadatel neodpoví, nicméně spojení zůstane po nějaký čas otevřené. Při velkém množství nových TCP spojení dojde k vyčerpání dostupné paměti.

Rovněž skenování sítě a zjišťování dostupných služeb je z hlediska správy cache velmi rizikový provoz. Grafy na obr. 7.1 zobrazují průběh počtu odstraněných toků (výpadků) za sekundu a úspěšnost vyhledání stavů v cache na vzorku dat s anomálií. Velikost cache je nastavena na 8192 stavů, správa paměti je zvolena LRU.

Ve 125. sekundě dochází k průdkému nárůstu počtu toků odstraněných z cache toků a zároveň snížení úspěšnosti vyhledání stavů v cache pod hranici 50%. V tomto případě se jedná o skenování sítě za účelem zjistit, na kterých počítačích je dostupná služba Microsoft SQL server naslouchající na portu 1433 a 1434. Samotný provoz byl odeslán z počítače napadaným červem *SQL Slammer worm*. Po odhalení MS-SQL serverů se červ pokusí proniknout do systémů pomocí využití chyby v implementaci MS-SQL. Tato chyba dovolí útočníkovi přeplnit zásobník a způsobí vykonání vloženého zdrojového kódu. Přestože samotný průchod sítí a zjišťování dostupných služeb nelze považovat za útok v pravém slova smyslu, může způsobit selhání správy cache. Výše uvedené aktivity zaplní cache stavy krátkých toků. Tyto stavy ale nemají žádný přínos z pohledu stavového zpracování toků,



Obrázek 7.1: Anomálie způsobená skenováním sítě ve vzorku dat *Mawi-2009/03/30-00:00*.

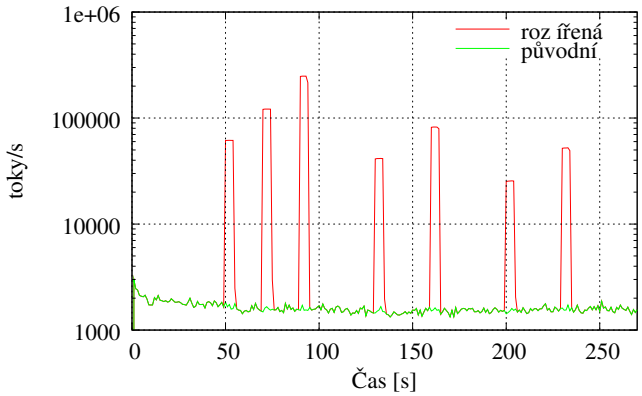
neboť se typicky jedná o toky s několika pakety, nejčastěji jedním, u nichž není udržení stavu toku důležité. Jiang et. al. [34] poukazuje na to, že v těchto případech dochází k významnému snížení propustnosti síťového zařízení a zvýšení latence zpracování paketů. Přestože anomálie představuje pouze část provozu, který je směřován na určitou službu a server v síti, bývá nepřímo postižen i zbylý provoz procházející společně s anomálií síťovými zařízeními. Z tohoto důvodu by správa cache měla chránit stavy toků náležící běžnému provozu, zatímco stavy náležící anomálii by měly být z cache rychle odstraněny.

V této práci byla navržena správa cache pro sledování stavů velkých toků, GARP-V2. Tato správa má velmi vysokou úspěšnost vyhledání stavu velkých toků, nicméně celková úspěšnost vyhledání stavů v cache je nižší než u jiných správ paměti jako GARP, SLRU-13, LRU. V průběhu anomálie by ale správa GARP-V2 mohla ochránit stavy existujících velkých toků, u kterých je vysoká pravděpodobnost, že k anomálii nenáleží. Těchto toků je malé množství, zároveň ale přenáší většinu dat, která nebudou ovlivněna negativními projevy anomálie. Výhodu tohoto přístupu lze předvést na příkladu OpenFlow přepínače. Pokud OpenFlow přepínač neví kam přeposílat pakety daného toku, dotáže se OpenFlow kontroléru. Na základě odpovědi od OpenFlow kontroléru instaluje přepínač do své cache stav obsahující směrovací informace. Při záplavě nových toků je kontrolér zahlcen požadavky a doba jeho odpovědi může být velmi dlouhá. Pokud navíc dojde ke ztrátě stavu velkého toku, pak služba využívající tento tok může zaznamenat výpadky či ztrátu celého spojení.

7.1 Rozšíření datové sady o anomálie

Pro analýzu chování správ cache je použita základní datová sada *Mawi-2010/04/14-14:00*. Vlastnosti této datové sady byly popsány v kapitole 3.4. Samotná sada neobsahuje žádné zjevné anomálie. Proto je uměle rozšířena o anomálie různých velikostí a typů. Rozšířená datová sada *Mawi-2010/04/14-14:00−anom* dovoluje provést řízené experimenty, v nichž je možné přesně sledovat chování správ v reakci na vložené anomálie. Obrázek 7.2 zachycuje průběh počtu toků za sekundu v původní sadě *Mawi-2010/04/14-14:00* a v rozšířené sadě *Mawi-2010/04/14-14:00−anom*.

Do původního provozu jsou přidány pakety způsobující prudký nárůst nových toků během omezeného intervalu. Do sady je přidáno několik takových intervalů. Každý interval trvá 5 sekund a simuluje útoky nebo skenování různých velikostí a typů. Během prvních tří intervalů je simulován útok záplavou nových TCP SYN paketů směřujících na port



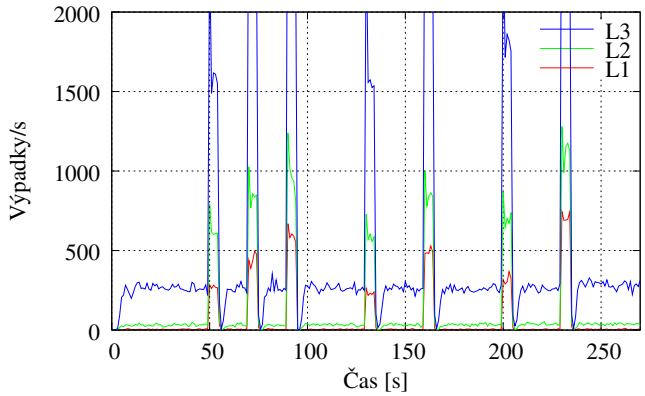
Obrázek 7.2: Počet toků za sekundu v původní a rozšířené datové sadě *Mawi-2010/04/14-14:00−anom*.

80 z několika zdrojů a náhodně generovaným zdrojovým číslem portu. Jedná se o toky s pouze jedním paketem. Počet toků za sekundu vygenerovaných v každém intervalu postupně vzrůstá z 60 tisíc toků/s na 120 tisíc toků/s a 240 tisíc toků/s v čase 50 s, 70 s, 90 s. Při generování největší záplavy je dosažena plná saturace linky. V časech 130 s a 160 s je simulován útok na aplikační úrovni, kdy každý tok obsahuje tři pakety. To znamená, že je simulováno ustavení TCP spojení a následné odeslání aplikačního požadavku na získání dat (například dotaz na SQL databázi). Získání dat stojí výpočetní zdroje. Velký počet požadavků zahltí výpočetní kapacitu napadeného systému, který přestane reagovat na oprávněné požadavky. Čas mezi sousedními pakety v toku je stanoven na 20 ms, aby bylo simulováno zpoždění při navazování spojení a posílání požadavků. Interval začínající ve 130 s obsahuje provoz se záplavou 40 tisíc toků/s a interval začínající ve 160 s obsahuje provoz se záplavou 80 tisíc toků/s. Poslední dva intervaly v rozšířené datové sadě simulují záplavu nových toků s pěti pakety na tok. Počet vygenerovaných toků/s je 25 tisíc, respektive 50 tisíc pro intervaly začínající v 200 s a 230 s. Časový úsek mezi intervaly obsahující záplavy s jednopaketovými toky je stanoven na 20 s a u záplav s toky s více pakety je stanoven na 30 s. Dostatečný časový rozestup mezi intervaly záplav dovoluje cache vrátit se do stabilního stavu před další záplavou.

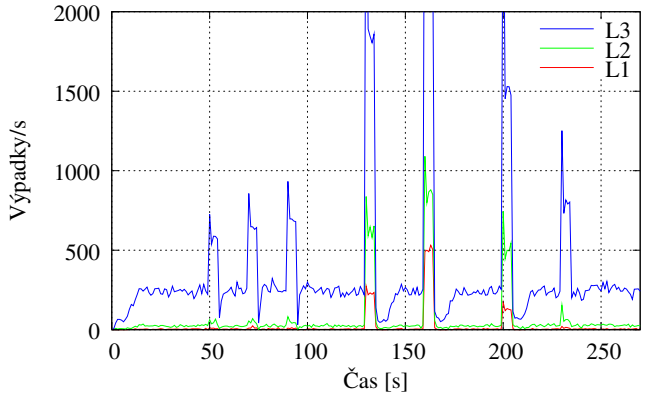
7.2 Hodnocení správ

V této kapitole je u vybraných správ cache zkoumáno chování během simulace správy cache na rozšířené datové sadě *Mawi-2010/04/14-14:00−anom*. Velikost cache je nastavena na velikost 8192 stavů toků. Pro analýzu byly vybrány správy hodnocené v předchozí kapitole, tj. LRU, SLRU, LIRS, S^3-LRU-7, LFU*-aging a GARP-V2. Správa LRU a SLRU-13 pracuje při běžném síťovém provozu velmi blízko dosažitelného optima, jak lze vidět z výsledků v tabulce 6.3. Správa paměti LIRS byla navržena pro odstranění potíží správ založených na LRU čelit záplavě nových položek v cache. Ze skupiny správ založených na sledování počtu přístupů se nejlépe osvědčila LFU*-aging. S^3-LRU-7 je správa paměti navržena pro sledování velkých toků v cache. Nakonec je porovnáno chování vyvinuté správy GARP-V2 a optimální správy FITF-V. FITF-V je modifikace FITF za účelem udržování stavů velkých toků kategorií L_1, L_2, L_3.

Správa LRU je určitým způsobem implementována ve všech zařízeních pracujích na základě neaktivního intervalu. Za běžných podmínek podává LRU dobré výsledky, nicméně



Obrázek 7.3: Chování správy LRU na rozšířené datové sadě *Mawi-2010/04/14-14:00--anom*.

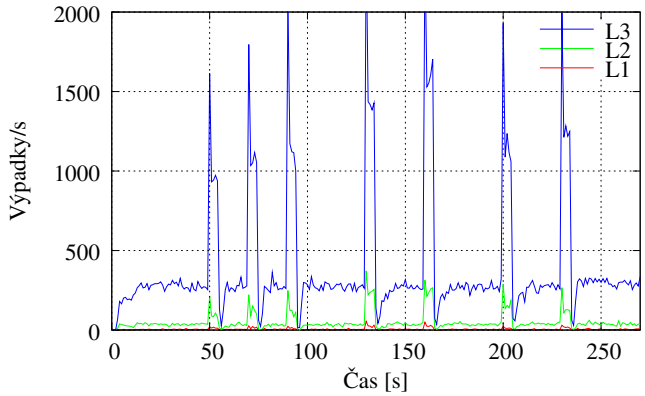


Obrázek 7.4: Chování správy SLRU-21 na rozšířené datové sadě *Mawi-2010/04/14-14:00--anom*.

selhává v udržení stavů velkých toků v cache, pokud se objeví velké množství nových toků s malým množstvím paketů. Obrázek 7.3 zobrazuje grafy počtu výpadků v jednotlivých kategoriích velkých toků. Lze pozorovat, že již při první záplavě nových toků s malou intenzitou dochází k mnoha výpadkům stavů velkých toků ve všech kategoriích. Primárním důvodem tohoto chování je pozice vkládání nově příchozích toků na začátek řádku. Nově příchozí toky tak získávají větší důležitost než stávající toky. Stávající toky jsou odsouvány směrem ke konci řádku, čímž se zvyšuje pravděpodobnost jejich odstranění z cache. Během záplavy cache spravované pomocí LRU dochází nevyhnutelně k odstranění všech stavů toků, které obdrží méně paketů za sekundu než je počet nových toků.

SLRU odstraňuje tento nedostatek posunutím pozice pro vkládání nových toků do vnitřní části řádku. Řádek je tak rozdělen na dvě části, chráněnou a nechráněnou. Chráněná čast řádku nalevo od pozice vkládání zajišťuje ochranu stávajícím tokům před nově vznikajícími toky. Do chráněné části se mohou dostat pouze toky, které obdrží více než jeden paket během jejich výskytu v nechráněné části řádku napravo od pozice vkládání. Velikost chráněné části musí být dost velká na uchování stavu velkých toků. Při experimentech v předešlé kapitole bylo zjištěno, že vkládání nových toků na konec druhé třetiny řádku je vhodné pro udržení velkých toků v cache (odtud SLRU-21). Stavy toků v nechráněné části soupeří s nově příchozími toky náležící záplavě.

Obrázek 7.4 ukazuje schopnost SLRU-21 vypořádat se s jedno-paketovými toky během



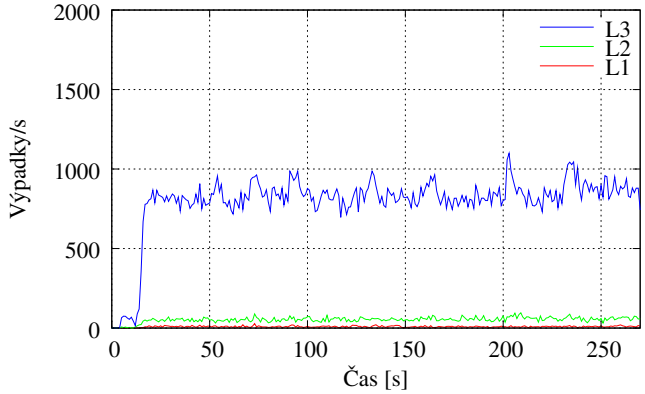
Obrázek 7.5: Chování správy S^3-LRU-7 na rozšířené datové sadě *Mawi-2010/04/14-14:00–-anom*.

prvních třech period záplav. Pokud ale toky v záplavě obsahují více paketků, pak SLRU-21 odstraňuje stavy velkých toků ve srovnatelné míře jako LRU. Důvodem je velmi jednoduchá strategie pro správu stávajících stavů v řádku. Pokud stav nového toku obdrží alespoň jeden další paket, je stav přesunut na začátek řádku. Tím jsou odsouvány stavy velkých toků do nechráněné části, kde soupeří o místo s toky záplavy.

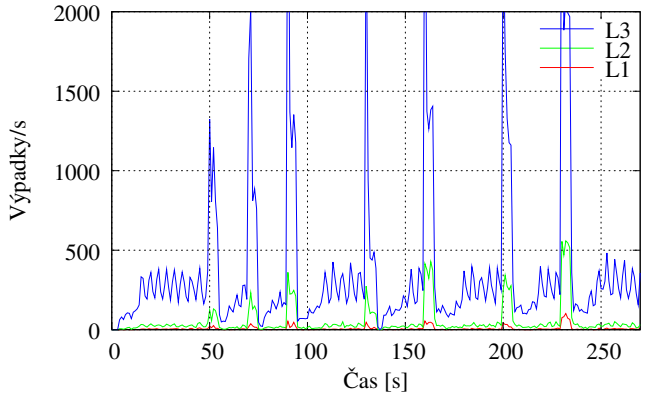
Za účelem zlepšení správy chráněné i nechráněné části byla navržena autorem této práce spolu s jeho kolegy správa S^3-LRU-7 [64]. Na rozdíl od SLRU neřadí S^3-LRU-7 stavy toků podle jejich posledního přístupu. S^3-LRU-7 posouvá stav toku pouze o jednu pozici směrem k začátku řádku, pokud je stav toku aktualizován. Dochází tak k prohození pořadí aktualizovaného stavu a jeho levého souseda. Jako důsledek je mnohem pravděpodobnější, že v chráněné části spolu budou navzájem zápasit pouze stavy velkých toků. Při experimentech bylo zjištěno, že pro S^3-LRU-7 nejlépe uchovává stavy velkých toků, pokud vstupní pozice je nastavena do první třetiny řádku (konkrétně na pozici 7). Grafy na obrázku 7.5 ukazují nízký počet odstranění stavů kategorie L_1 během záplavy nových toků v porovnání s ostatními správami. Na druhou stranu stavy velkých toků z kategorií L_2 a L_3 jsou odstraňovány ve zvýšeném množství.

Další testovanou správou je LIRS. Správa LIRS vyžaduje nastavení parametru, který určuje velikosti seznamů A_m a A_1. Autoři správy LIRS doporučují nastavit velikost A_m na 99% velikosti cache toků a zbytek ponechat pro A_1. V předchozích experimentech se síťovým provozem a LIRS správou se ukázalo jako vhodnější nastavit velikost A_m na 90%. Na obrázku 7.6 je vidět chování LIRS při simulaci s nastavením A_m na 7300 stavů a A_1 na 892 stavů. Je vidět, že s tímto nastavením je chování LIRS velmi stabilní. Periody záplav nových toků nemají vliv na počet odstraněných stavů ve velkých kategoriích L_1, L_2, L_3. Při bližším zkoumání lze ale pozorovat zvýšený počet odstraněných stavů velkých toků v průběhu celé simulace. Počet odstraněných stavů velkých toků mezi periodami záplav je přibližně dvojnásobný v porovnání s počtem odstraněných stavů předchozími správami. Tento stav je nežádoucí, neboť nevede ke snižování výpadků stavů z cache.

Správa LFU*-aging vyžaduje nastavení dvou parametrů, M_{ref} omezující maximální počet počítaných přístupů a A_{max} zajišťující stárnutí uložených stavů pomocí zmenšení hodnoty čítačů na polovinu po každé po A_{max} přístupech do cache. V provedených experimentech bylo zjištěno, že pro udržení stavů velkých toků v cache je vhodné nastavit $M_{ref} = 10$ a $A_{max} = 100000$. Chování správy na rozšířené datové sadě *Mawi-2010/04/14-14:00–anom* je zachyceno na obr. 7.7. LFU*-aging se snaží o zachování stávajících toků i během záplav



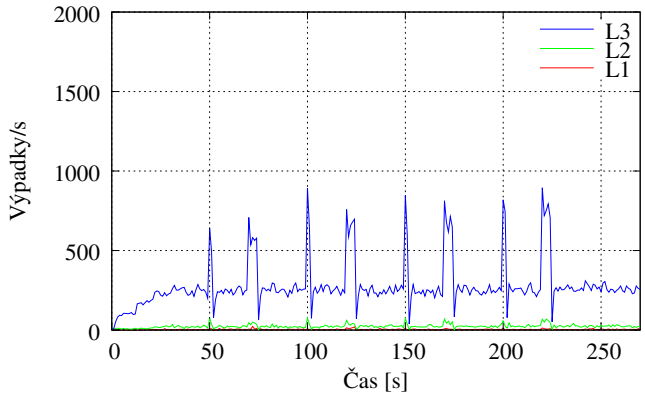
Obrázek 7.6: Chování správy LIRS na rozšířené datové sadě *Mawi-2010/04/14-14:00–-anom*.



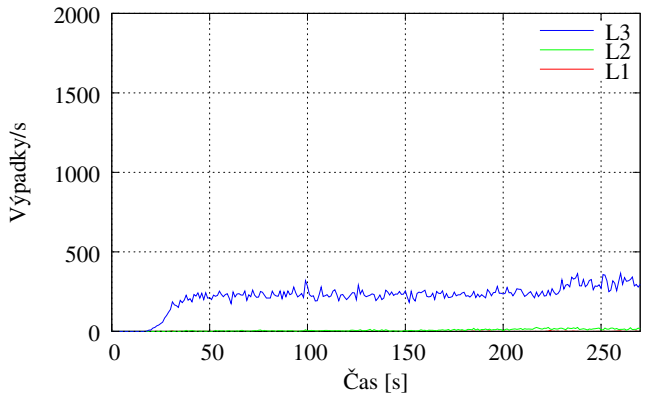
Obrázek 7.7: Chování správy LFU*-aging na rozšířené datové sadě *Mawi-2010/04/14-14:00–anom*.

díky počtu čítačů předchozích přístupů. Přesto dochází v době záplav k výpadkům stavů velkých toků přibližně ve stejné míře jako u správy S^3-LRU-7. Obě správy si uchovávají historii nedávných přístupů (S^3-LRU-7 v podobě pozice v řádku, LFU v podobě čítače). Důvodem proč LFU*-aging není schopna plně odolat záplavám je fixní nastavení parametrů $M_{ref} = 10$ a A_{max}, které je uzpůsobeno na běžný provoz. V době záplav by bylo vhodné tyto parametry přizpůsobit aktuální situaci, například výrazně zamezit stárnutí existujících stavů v cache. Periodické kolísání počtu výpadků během běžného provozu odpovídá periodickému spoustění procesu stárnutí. Kolísání je dobře viditelné především na kategorii toků L_3.

Předposlední testovanou správou je GARP-V2 z obrázku 6.12 pro udržování stavů velkých toků v cache. Zaměření této správy na velké toky dovoluje GARP-V2 přečkat záplavu krátkých toků bez vážného narušení průběhu velkých toků. Na rozdíl od LIRS uchovává stavy velkých toků bez zvýšeného počtu výpadků i za běžného provozu a nejvíce se tak přibližuje ideální správě paměti FITF-V. V porovnání s S^3-LRU-7 poskytuje GARP-V2 optimalizovaný vektor přesunů, který zajišťuje rychlý přesun stavů velkých toků do chráněné části řádku. V chráněné části řádku dochází pouze k velmi pozvolnému posouvání přistupovaných stavů směrem k začátku. Díky tomu může GARP-V2 přestát i záplavy toků, které obsahují více paketů. Obrázek 7.8 ukazuje velmi stabilní chování GARP-V2 v průběhu



Obrázek 7.8: Chování správy GARP-V2 na rozšířené datové sadě *Mawi-2010/04/14-14:00−anom*.



Obrázek 7.9: Chování správy FITF-V na rozšířené datové sadě *Mawi-2010/04/14-14:00−-anom*.

celého simulovaného intervalu.

Teoretický limit představuje správa FITF-V. V datové sadě *Mawi-2010/04/14-14:00−-anom* se vyskytuje více než 8192 souběžných toků kategorie L_1, L_2, L_3. Proto i ideální správa FITF-V generuje určitý počet výpadků. Vzhledem ke znalosti datové sady nemohou mít periody záplav na FITF-V nejmenší vliv, jak lze pozorovat na obrázku 7.9.

Tabulka 7.1 vzájemně porovnává snížení počtu výpadků správ zaměřených na velké toky na třech nejintenzivnějších záplavách. Porovnání zachycuje snížení počtu výpadků dané správy vůči základní správě LRU během záplav. Porovnání je vyjádřeno jako procentuální poměr maxim počtu výpadků za sekundu vůči výsledkům LRU. Maximum počtu výpadků za sekundu je vybráno z časového úseku, který začíná v době začátku záplavy a končí 5 sekund po skončení záplavy. Charakteristika „maximum počtu výpadků" je zvolena ze dvou důvodů. Odráží nejhorší situaci, která nastane během záplavy. Je jednoduché toto maximum určit na rozdíl od součtu všech výpadků způsobených záplavou. U součtu nelze dobře určit, zda se již jedná o výpadky způsobené záplavou nebo běžným provozem. Každá správa se chová odlišně a reakce na záplavu mohou mít různě dlouhý projev.

Chování správ zachycené na předchozích obrázcích a výsledky v tabulce 7.1 ukazují, že vyvinutá správa GARP-V2 odolává záplavám nových toků lépe než ostatní správy. V době záplav je počet výpadků srovnatelný se správou LIRS, která pracuje nezávisle na příchozích

Tabulka 7.1: Snížení počtu výpadků vzhledem k správě LRU (počet toků záplavy *f*, počet paketů na tok *p*, trvání *t*, LIRS99 alokuje 99% cache pro stavy označené LIR, zatímco LIRS90 pouze 90%)

f=240 tis. toků/s, p=1, t=5 s			
Správa	L_1	L_2	L_3
LRU	610	995	2600
SLRU-21	2.1%	6.8%	27.5%
S^3-LRU-7	3.0%	18.7%	61.4%
GARP-V2	2.3%	5.9%	32.7%
LIRS90	3.4%	7.4%	35.8%
LIRS99	3.3%	7.4%	37.9%
FITF-V	0.2%	1.0%	10.2%

f=40 tis. toků/s, p=3, t=5 s			
Správa	L_1	L_2	L_3
LRU	243	581	1582
SLRU-21	39.0%	63.5%	104.9%
S^3-LRU-7	6.7%	25.0%	70.9%
LIRS90	38.5%	80.0%	124.9%
LIRS99	2.8%	6.7%	37.9%
GARP-V2	3.3%	13.2%	62.0%
FITF-V	0.3%	0.9%	9.2%

f=25 tis. toků/s, p=5, t=5 s			
Správa	L_1	L_2	L_3
LRU	750	1188	2729
SLRU-21	3.1%	13.7%	38.7%
S^3-LRU-7	3.9%	22.4%	56.7%
LIRS90	2.3%	8.5%	38.3%
LIRS99	2.5%	9.6%	42.3%
GARP-V2	1.6%	8.2%	35.3%
FITF-V	0.2%	2.3%	11.0%

záplavách, nicméně odstraňuje mnoho stavů velkých toků i během výskytu běžného provozu. GARP-V2 v obdobích běžného provozu naopak minimalizuje množství odstraněných stavů velkých toků. Z hlediska snížení celkového počtu výpadků dopadlo nejlépe GARP-V2 na sadě *Mawi-2010/04/14-14:00−anom*. Pokud je ale vyvinutá GARP-V2 hodnocena na běžném vzorku dat *Mawi-14:15*, pak dosahuje horších výsledků než SLRU-13. Zatímco SLRU-13 dosahuje přibližně 6% pravděpodobnosti výpadků p_ν, správa GARP-V2 dosahuje úspěšnosti 7%.

Kapitola 8

Rozšíření správy

Doposud zkoumané a testované správy cache byly založeny pouze na uchovávání historie předcházejích přístupů. Tato historie byla uchovávána formou pořadí v seznamu (v případě GARP, LRU, SLRU) nebo explicitně jako klouzavé průměry u EXP1, LFU-aging. Porovnání těchto správ s teoretickou správou FITF ukázalo, že existuje velký prostor pro zlepšení. Nicméně ani optimalizace správy na velikost cache a datovou sadu pomocí GA není schopna plně tohoto prostoru využít. Na základě těchto pozorování lze usuzovat, že množství informace uchované v historii předchozích přístupů je z hlediska předpovědi následujícího přístupu vyčerpáno.

Další informaci, která může sloužit pro rozhodování o uchování nebo odsunutí toku, lze hledat v samotných paketech každého toku. Pakety typicky obsahují záhlaví různých protokolů. Některá z polí v záhlaví mohou obsahovat příznaky vývoje toku nebo příznaky určitých událostí, které se vztahují k jeho sestavení a řízení. Z pohledu zpracování síťových toků má smysl zkoumat příznaky vyskytující se na síťové a transportní vrstvě. Vyšší vrstvy síťová zařízení typicky nezpracovávájí, přestože sémantika dat vyšší vrstvy by byla pravděpodobně dobrou nápovědou pro budoucí vývoj toku. Například konec zprávy protokolu SMTP lze rozpoznat na základě detekce sekvence znaků „CRLF.CRLF“ (tečka ohraničena dvěma prázdnými řádky).

Obecně se nelze na příznaky plně spoléhat, neboť některé příznaky se vyskytují pouze u některých protokolů, například pouze u TCP. Navíc některé pakety mohou být doručeny jinou cestou než přes dané zařízení nebo mohou být ztraceny (na opakované zaslání se nelze spoléhat) nebo mohou být záměrně neposlány jak je tomu v případě některých útoku (v takovém případě je TCP spojení ukončeno až na základě vypršení intervalu pro neaktivní tok).

Rozšíření správy cache pak spočívá ve vyřešení dvou problémů. Nejprve je třeba nalézt vhodné příznaky paketů, které mohou mít pozitivní vliv na správné rozhodování při správě toků. Následně je třeba navrhnout způsob kombinace nalezených příznaků se správou cache.

K vyřešení prvního problému provedeme analýzu závislosti velikosti vzdálenosti mezi pakety toku na příznacích jednotlivých paketů. Cílem je zjistit, zda hodnota některého příznaku může s určitou pravděpodobností naznačit nadcházející velikost vzdálenosti zkoumaného paketu od následujícího paketu stejného toku a napodobit tak fungování FITF. Souběžně provedeme analýzu závislosti velikosti toku na příznacích paketů, kdy je cílem zjistit, zda hodnoty některého příznaků mohou pomoci určit velké toky.

Pozornost je zaměřena na pole v záhlaví paketu, která mohou mít nějaký vztah k vývoji toku. Pole, která tento vztah nemají, nemá smysl analyzovat. Mezi pole, která budou analyzována, patří:

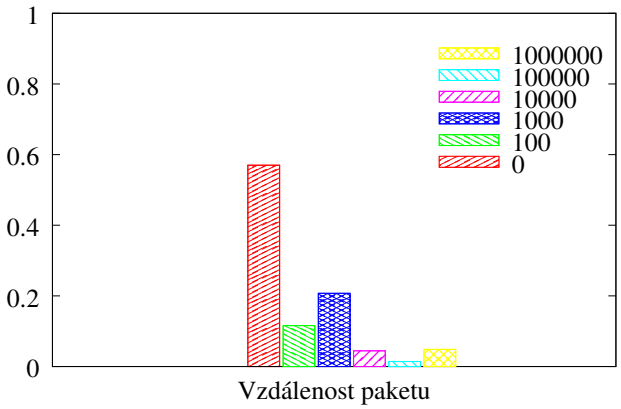
- IP protocol (v záhlaví IP protokolu),
- Type of Service (v záhlaví IP protokolu),
- Total Length (v záhlaví IP protokolu),
- Fragment Flags (v záhlaví IP protokolu),
- TCP Flags (v záhlaví TCP protokolu),
- TCP Window (v záhlaví TCP protokolu),
- TCP Source port (v záhlaví TCP/UDP protokolu) a
- TCP Destination port (v záhlaví TCP/UDP protokolu).

8.1 Příznaky a vzdálenost následujícího paketu

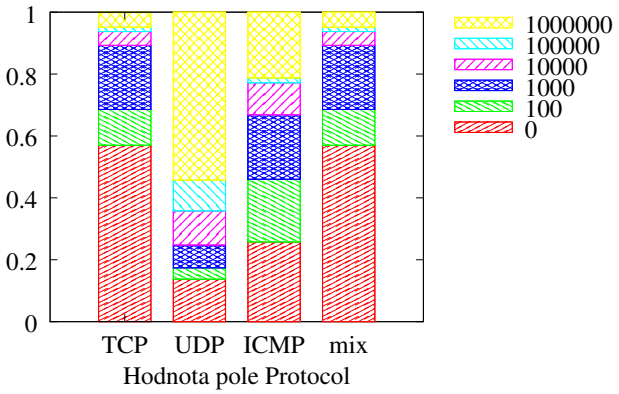
Nejprve je u těchto polí provedena analýza závislosti následující vzdálenosti paketu stejného toku na hodnotě pole aktuálního paketu. Analýza je založena na porovnání rozložení (histogramů) vzdáleností následujících paketů pro různé hodnoty polí v záhlaví aktuálního paketu. Pokud se tvar histogramu liší od tvaru histogramu pro další hodnoty pole, resp. od tvaru histogramu pro rozložení vzdáleností všech paketů bez ohledu na záhlaví, pak lze říci, že daná hodnota poskytuje určitou informaci pro odhad následující vzdálenosti. U paketů, které neobsahují některá z polí (například UDP pakety neobsahují některá TCP pole) mají tato pole nedefinovanou hodnotu a nejsou pro účely analýzy započítávána. To je ve shodě s následným využitím polí při klasifikaci, neboť samotné klasifikaci předchází fáze zpracování záhlaví paketu, která určí platná pole v záhlaví. Pro účely této analýzy je využita datová sada *Mawi-2010/04/14-14:00*. Data každého paketu v těchto sadách musí být rozšířena o informaci o vzdálenosti příchodu dalšího paketu stejného toku. Vzdálenost dalšího paketu stejného toku je měřena v počtech paketů ostatních toků. Dochází tak ke ztrátě časové informace, ta je ale pro samotné fungování cache nepodstatná. Podstatné jsou pouze příchody paketů. Obrázek 8.1 zobrazuje histogram těchto vzdáleností pro všechny pakety. Histogram je normalizovaný celkovým počtem paketů. Vzdálenosti jsou rozděleny do několika intervalů: $\langle 0, 100)$, $\langle 100, 1000)$, $\langle 1000, 10000)$, $\langle 10000, 100000)$, $\langle 100000, 1000000)$ a $\langle 1000000, \infty)$. V obrázku je každý interval označen pouze svou spodní hranicí. Poslední interval v sobě zahrnuje nekonečnou vzdálenost, která značí, že daný paket byl poslední daného toku a žádný nadcházející paket pro daný tok v datové sadě neexistuje. Vzhledem k tomu, že datová sada je časově omezena, pak určitému počtu paketů je nekonečno přiřazeno nesprávně, neboť paket pro stejný tok by přišel v následujícím časovém intervalu. Díky velikosti datové sady je ale takových paketů méně než 1‰, což bylo empiricky ověřeno na několika datových sadách.

Prvním zkoumaným polem je IP protocol. Toto pole určuje, jaký protokol se nachází v následující vrstvě. Z datových sad bylo zjištěno, že nejčastější je využití protokolu TCP (80% paketů) následuje UDP (15% paketů), tunelovaný provoz a ICMP (přibližně 2% paketů). Ostatním protokolům připadá výrazně méně provozu.

Z obrázku 8.2 lze pozorovat, že rozložení vzdáleností následujících paketů u protokolu TCP nejvíce odpovídá rozložení pro veškerý provoz. Tento stav odráží většinové zastoupení TCP protokolu v provozu. U protokolu UDP lze pozorovat posun směrem ke dlouhým vzdálenostem. Tento posun je způsoben tím, že většina UDP toků jsou toky obsahující



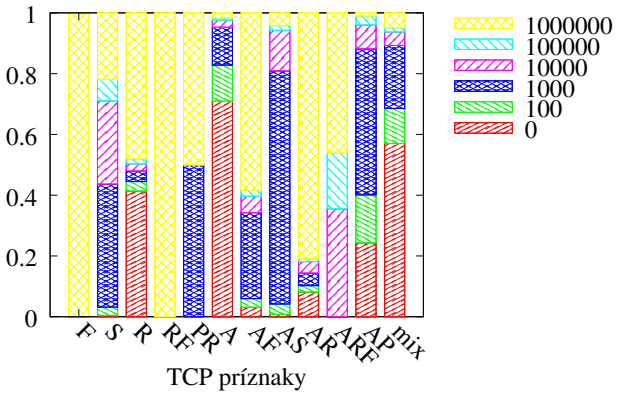
Obrázek 8.1: Histogram rozložení vzdáleností paketů stejného toku.



Obrázek 8.2: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro různé protokoly.

pouze malé množství paketů. U protokolu ICMP se nevyskytují transportní porty. Tok ICMP paketů je určen pouze na základě protokolu a IP adres. Lze vidět, že v ICMP tocích mají pakety delší vzdálenosti mezi sebou než u veškerého provozu. Na základě pole IP protocol lze tedy dobře předpovídat delší vzdálenost, pokud se ve vyšší vrstvě nachází protokol UDP případně ICMP.

Nejdůležitějším příznakem pro TCP toky bude s největší pravděpodobností obsah pole TCP flags. Toto pole obsahuje bity, které navazují, řídí a ukončují dané TCP spojení. Bit nazvaný SYN vyjadřuje žádost na ustavení spojení. Protistrana odpovídá na tento požadavek nastavením bitů SYN a ACK. Následuje spolehlivý přenos dat. Během tohoto přenosu je nastaven bit ACK vyjadřující potvrzení doručení přenášených dat. Rovněž může být nastaven bit PSH. Tento bit signalizuje transportní vrstvě, že data z takto označeného paketu nemají být ukládána do vyrovnávací paměti, ale mají být přímo zaslána protistraně. Protistrana rovněž nemá ukládat tato data do vyrovnávací paměti. Ukládání dat do vyrovnávací paměti slouží pro zajištění efektivity přenosu z hlediska využití šířky pásma. Důvodem k nastavení tohoto příznaku je minimalizace zpoždění zasílaných dat. Pokud odesílající aplikace ví, že aktuálně již nemá žádná další data, nastaví tento bit, aby data čekající ve vyrovnávací paměti byla okamžitě odeslána. Typicky se tak děje u vzdáleného přístupu k počítači, například pomocí SSH. U takové aplikace není většinou uživatel schopen vygenerovat takové množství dat, aby zaplnilo vyrovnávací paměť, a tím pádem byla tato data odeslána.



Obrázek 8.3: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro různé kombinace nastavení pole TCP flags.

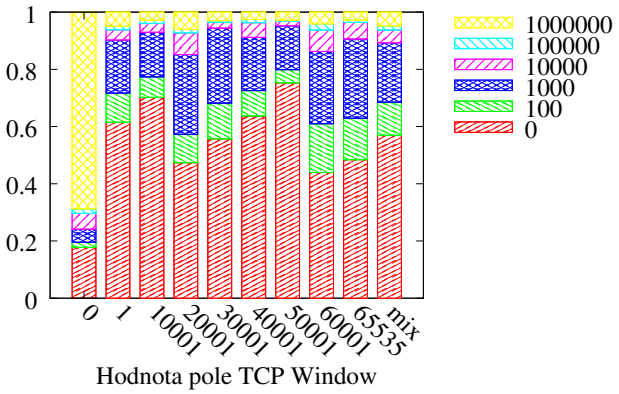
Zároveň uživatel očekává odezvu SSH konzole v reálném čase. Proto většina paketů bude mít nastavena bit PSH, aby nedocházelo ke zpoždění přenášených dat ve vyrovnávacích pamětech. Bit URG se v dnešní době nepoužívá a nebyl nastaven ani v jednom paketu. Pokud spojení není aktivní nebo dojde k chybě, pak je zaslán paket s nastaveným bitem RST, který signalizuje pokus o navrácení do výchozí stavu, ze kterého by mohlo být spojení znovu navázáno. Spojení je běžně ukončeno výměnou paketů s nastaveným bitem FIN.

Určitý význam může mít sledování nastavení jednotlivých bitů. Nicméně je vhodnější pozorovat přípustné kombinace nastavených bitů a interpretovat tak význam jednotlivých bitů v kontextu ostatních. Obrázek 8.3 zobrazuje pro každou kombinaci, která se v datové sadě nacházela, rozložení vzdáleností nadcházejících paketů. Na vodorovné ose jsou vyneseny jednotlivé kombinace TCP příznaků, zatímco svisle jsou zobrazeny histogramy počtu vzdáleností vytvořené dle intervalů definovaných výše. Každý histogram je normalizován počtem paketů obsahující danou kombinaci TCP příznaků. Délka obdelníku tak vyjadřuje poměr mezi zastoupením jednotlivých intervalů a součet délek obdelníků jednoho histogramu je roven jedné. Jednotlivé názvy TCP bitů byly zkráceny na první písmeno jejich pojmenování (například S = SYN) kombinace nastavených bitů je vyjádřena složením jejich zkratek (například ARF = ACK + RST + FIN). Rozložení všech vzdáleností paketů stejného toku bez ohledu na nastavené příznaky je vyjádřeno posledním sloupcem označeným na x-ové ose jako *mix*.

Z obrázku 8.3 je vidět, že existují kombinace nastavených bitů, které dokáží velmi přesně napovědět, jakou bude mít následující paket daného toku vzdálenost od současného. Například kombinace FIN, RST+FIN, ACK+RST+FIN označují konec přenosu, po kterém se již žádný další paket neobjevil. Rozložení vzdáleností mezi prvním paketem toku s nastaveným SYN a následujícím paketem se značně liší od běžného rozložení vzdáleností všech paketů. Tato prodleva je způsobena navazováním spojení, kdy před odesláním dalšího paketu je nutné počkat na odpověď druhé strany. Zároveň se nemusí podařit spojení navázat a není tak možné zaslat další pakety. Rozložení vzdáleností je posunuto k delším vzdálenostem, s malým zastoupením krátkých vzdáleností. Podobný průběh mají i kombinace ACK+FIN, ACK+RST, PSH+RST. Posun k delším vzdálenostem je způsoben záměrem ukončit spojení. Zajímavé rozložení vzdáleností paketů předpovídá osamoceně nastavený bit RST. Zhruba polovina vzdáleností je velmi krátká a druhá polovina velmi dlouhá. Po prozkoumání paketů lze konstatovat, že krátké vzdálenosti jsou způsobeny zasláním dalšího paketu s RST bitem ve velmi krátké době po prvním RST paketu. Po druhém RST paketu již další paket

Tabulka 8.1: Zastoupení kombinací TCP flags v paketech (u kombinací zde neuvedených je zastoupení menší než 1‰).

Kombinace bitů	S	R	A	AF	AS	AR	AP	ARF
Zastoupení [%]	3,8	0,5	74,5	2,9	2,1	0,8	15,3	0,2



Obrázek 8.4: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro zvyšující se hodnotu TCP Window pole.

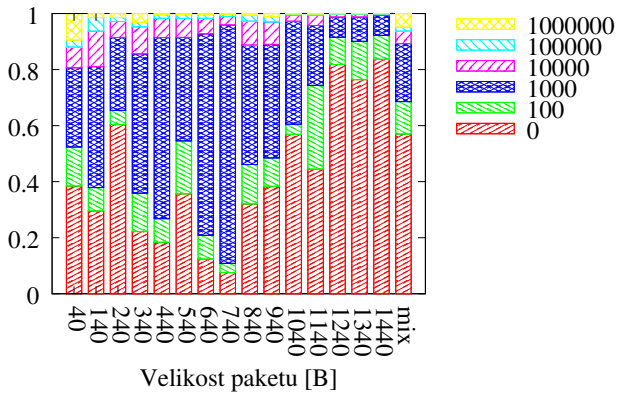
v převážné většině případů není zaslán. Rozložení vzdáleností pro osamoceně nastavený bit ACK se liší od běžného rozložení pouze částečně. Pravděpodobnost kratších vzdáleností po paketu s tímto nastaveným bitem je pravděpodobnější než u všech paketů. Bohužel většina paketů má nastaven právě pouze bit ACK, jak ukazuje tabulka 8.1. Je tedy těžké dobře předpovědět následující vzdálenost pro všechny pakety, neboť jasně určující kombinace pokrývají jenom malé procento paketů.

Dalším zkoumaným polem je TCP Window. Toto pole udává množství dat, které je ještě možné odeslat bez čekání na jejich potvrzení o přijetí. Smyslem je přizpůsobit přenos dat parametrům spojení. Velikost okna je zvětšována nebo zmenšována v závislosti na spolehlivosti a zpoždění doručení. Přizpůsobování okna provádí strana přijímající data. Pokud přijímající strana není schopna přijímat další data, informuje vysílající stranu pomocí nastavení TCP Window na hodnotu nula. Vysílající strana pak periodicky zasílá paket, který je druhou stranou potvrzen. Tím je zajištěno udržení spojení a navíc tak lze zjistit, zda je druhá strana znovu schopna příjmu. Různé hodnoty tohoto pole tak mohou naznačovat budoucí vývoj toku.

Obrázek 8.4 zobrazuje rozložení hodnot nadcházejících vzdáleností pro několik rozsahů hodnot pole TCP Window. První sloupec je určen pouze pro hodnotu 0, která má specifický význam v TCP komunikaci. Ostatní sloupce reprezentují histogramy pro velikost rozsahů 10000 až do hodnoty 65534. Sloupec s hodnotou 65535 reprezentuje histogram pouze pro tuto hodnotu. Histogram pro hodnotu 0 ukazuje významný posun v distribuci vzdáleností směrem k delším vzdálenostem. Po podrobném prozkoumání datových sad lze konstatovat, že po paketech s hodnotou TCP Window rovnu nule nastává buď přerušení komunikace (tj. velmi dlouhá vzdálenost) nebo velmi krátká vzdálenost, která je způsobena znovuzasláním stejného paketu. Rozložení histogramů pro rozsahy 1 až 60001 neukázalo žádný významný rozdíl oproti histogramu pro všechny vzdálenosti. Stejně tak histogram pro maximální hodnotu pole TCP Window (tj. 65535) je podobný histogramu pro všechny vzdálenosti a není

Tabulka 8.2: Rozložení množství hodnot pro rozsahy hodnot pole TCP Window.

Rozsahy	0	1	10001	20001	30001	40001	50001	60001	65535
Zastoupení [%]	1,2	40,0	14,5	1,8	10,7	1,2	3,1	11,1	16,4



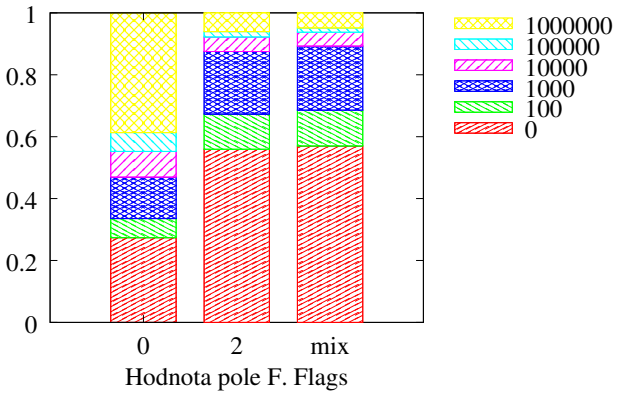
Obrázek 8.5: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro zvyšující se velikost paketu.

tak možné z tohoto pole usuzovat na vzdálenost následujícího paketu. Z pohledu klasifikace je tedy využitelná pouze hodnota nula. Z tabulky 8.1 je patrné, že množství paketů s hodnotou pole TCP Window rovnou nule je malé, což implikuje malý přínos tohoto pole pro odhad vzdálenosti následujícího paketu.

Ze záhlaví protokolu IP je dalším zkoumaným polem Total Length. Toto pole udává celkovou délku paketu. Dle RFC 791 [51] je vyžadováno, aby tato délka byla z důvodu malé velikosti vyrovnávacích pamětí v síťových zařízení omezena nejčastěji na velikost 1500 B. Z hlediska předpovědi následující vzdálenosti paketu je nutné si uvědomit, že pokud je k dispozici více dat pro odeslání, pak velikost paketů bude dosahovat právě horní hranice velikosti paketu a tyto pakety budou vysílány hned za sebou. Je velmi pravděpodobné, že velikost vzdálenosti těchto paketů bude s velkou pravděpodobností malá. Naopak u malých paketů lze očekávat, že množství dat na přenos není mnoho a nejsou tak při odvysílání malého paketu k dispozici. Vzdálenosti mezi pakety jsou tak větší. Rozložení vzdáleností následujících paketů stejného toku v závislosti na poli Total Length je zachyceno na obrázku 8.5.

Z obrázku 8.5 je patrné, že rozložení vzdálenosti pro pakety s velikostí nad 1240 B je posunuto ke kratším vzdálenostem a pravděpodobnost dlouhé vzdálenosti je velmi malá. U paketů s velikostí pod 140 B je tomu naopak. V porovnání s rozložením všech vzdáleností je více pravděpodobné, že po krátkém paketu následuje dlouhá vzdálenost k dalšímu paketu nebo že následuje konec toku. Zajímavé rozložení vzdáleností mají pakety se střední velikostí (kolem 700 B), kde pravděpodobnosti velmi krátké nebo velmi dlouhé vzdálenosti jsou velmi malé a nejvíce jsou zde zastoupeny vzdálenosti střední. Analýza těchto paketů ukázala, že převážná většina z nich náleží aplikaci SSH běžící na portu 22. Nicméně počet těchto paketů vůči celkovému počtu je malý. Rozložení zastoupení jednotlivých délek je bimodální s maximy kolem velikostí 60 B a 1480 B, jak ukazuje graf na obr. 3.4 v kapitole 3.4.

Pole ToS (Type of Service) v záhlaví protokolu IP je dle RFC 791 [51] určeno pro označení kvality služby, která je vyžadována daty přenášenými v daném paketu. Síťová zařízení po trase mohou upřednostnit na základě tohoto pole daný paket před ostatními



Obrázek 8.6: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro různá nastavení pole Fragment Flags.

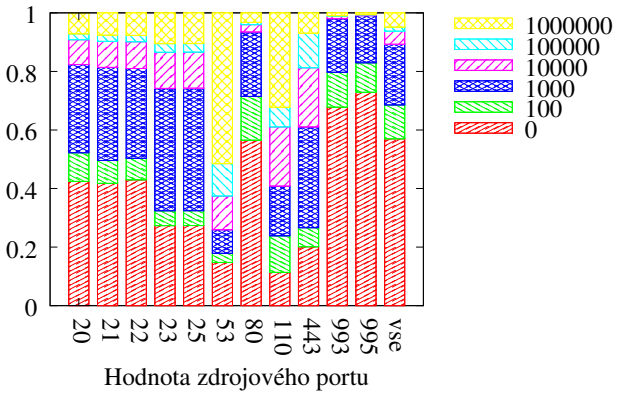
nebo ho naopak pozdržet ve svých vyrovnávacích pamětech než jsou odeslány pakety ostatní. Z hlediska predikce následující vzdálenosti paketu může být toto pole nápovědou o tom, jaká data jsou v paketu přenášena. Následně lze zjišťovat, zda lze dle určitého typu dat usuzovat na vzdálenost následujícího paketu. Bohužel pole ToS je prázdné u všech paketů a ve všech zkoumaných datových sadách. Z toho vyplývá, že toto pole není v současné době využíváné a není tak možné dle tohoto pole usuzovat na velikost následující vzdálenosti.

Dalším zkoumaným polem je pole Fragment Flags v záhlaví protokolu IP. Toto pole obsahuje bity, které značí, zda je daný paket možné rozdělit neboli fragmentovat a zda je fragmentovaný. Této vlastnosti se využívá v případech, kdy je třeba rozdělit paket z důvodu jeho značné délky, kterou nejsou schopny síť či zařízení po cestě zpracovat. Ve zkoumaných datových sadách se vyskytuje pouze malé množtví (méně než 1‰) fragmentovaných paketů (s hodnotou pole Fragments nastavenou na 1). To je v souladu i s obecným předpokladem, že v současné době není třeba dlouhé pakety fragmentovat, neboť zařízení mají dostatek kapacity pro zpracování dlouhých paketů. Zbývá tedy prozkoumat, zda hodnota pole rovna dvěma, která značí zákaz fragmentace paketu, má schopnost napovědět délku nadcházející vzdálenosti paketu.

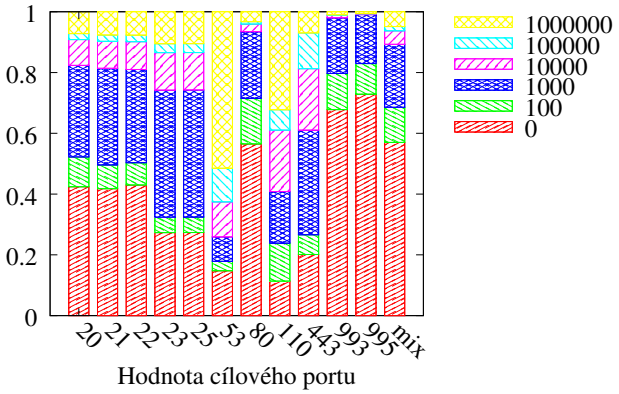
Z obrázku 8.6 je vidět, že rozložení vzdáleností paketů s různě nastavenými bity pro řízení fragmenetace je odlišné pro hodnotu nula. Tato hodnota zvyšuje pravděpodobnost výskytu delších vzdáleností, nicméně se vyskytuje pouze u přibližně 14% paketů. Částečně je tedy možné využít hodnotu pole Fragment Flags pro předpověď vzdálenosti následujícího paketu.

Data přenášena v tocích náleží ve většině případů určité aplikaci. Je možné, že různé aplikace mohou mít specifický vzor chování, co se týká vzdáleností mezi odesílanými pakety. Některé aplikace mohou generovat delší vzdálenosti než jiné, pokud například čekají na vstupy uživatele. Aplikace lze částečně rozpoznat podle zdrojového nebo cílového portu v transportní vrstvě. Na základě portu je možné identifikovat dobře známé aplikace běžící na vyhrazených portech do hodnoty 1024 (organizace IANA přiřazuje čísla portů vybraným aplikacím). Ostatní aplikace využívají zbylý rozsah náhodně nebo deterministicky. V analýze je pozornost zaměřena na dobře známé aplikace přenášející větší množství paketů.

Obrázky 8.7, 8.8 zobrazují rozložení vzdáleností paketů pro vybraná čísla zdrojových, respektive cílových portů. Rozložení pro vybrané aplikace je u obou histogramů velmi podobné. Bohužel rozložení vzdáleností portu 80, přes který je přenášeno velké množství dat,



Obrázek 8.7: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro vybrané zdrojové porty.

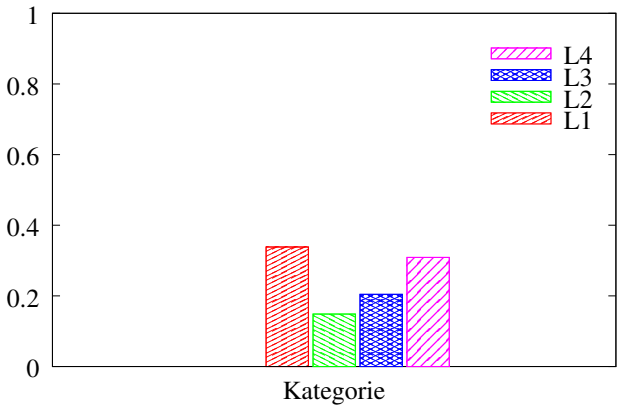


Obrázek 8.8: Sloupcové histogramy rozložení vzdáleností paketů stejného toku pro vybrané cílové porty.

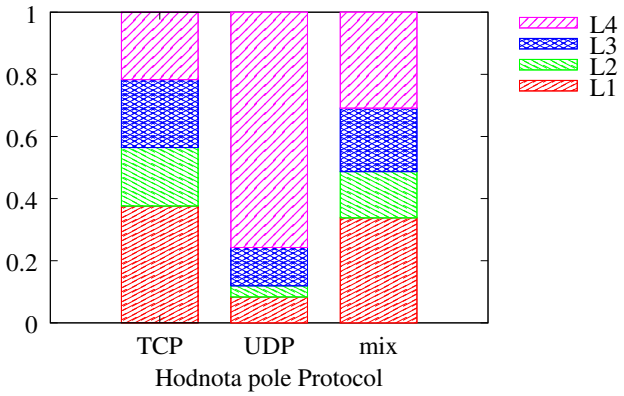
je velmi podobné s rozložením všech vzdáleností v provozu a nelze jej použít k predikci vzdáleností. Rovněž i ostatní porty mají velmi podobné rozložení až na port 53. Na portu 53 je typicky provozována služba překladu doménových jmen (DNS). DNS je ale ve většině případů přenášeno přes protokol UDP. Vzhledem k tomu, že DNS tvoří většinu provozu UDP, pak lze využít přímo pole IP protocol a není nutné využívat číslo portu. Z tohoto pohledu se jeví čísla portů transportního protokolu jako špatně využitelná pro predicki vzdáleností následujících paketů.

8.2 Příznaky a velké toky

Z hlediska predikce velikosti toku je analyzována stejná množina polí ze záhlaví síťové a transportní vrstvy jako u predikce vzdálenosti. Obrázek 8.1 zobrazuje histogram počtu paketů náležící k tokům v jednotlivých kategoriích normalizovaný počtem všech paketů. První tři kategorie L_1, L_2, L_3 v sobě obsahují přibližně 70% všech paketů a toky v nich obsažené jsou považovány za velké. Kategorie L_4 obsahuje toky s malým množstvím paketů. Vzhledem k velkému počtu malých toků v síťovém provozu je poměr paketů z kategorie L_4 na celkovém počtu paketů vysoký (přibližně 30%). Nejvíce se ovšem kategorie L_4 podílí na počtu toků, zabírá 90% a více.



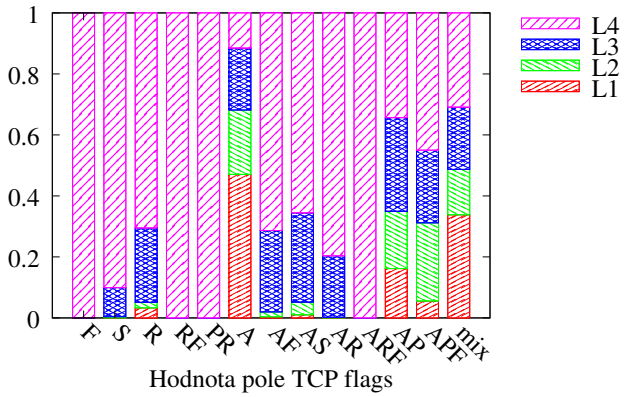
Obrázek 8.9: Histogram rozložení příslušnosti paketů do kategorií toků.



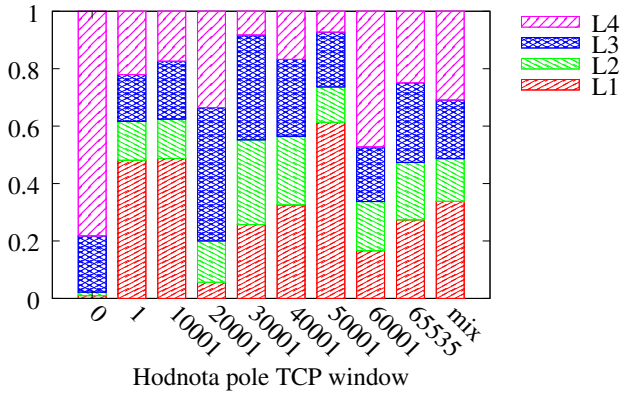
Obrázek 8.10: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků pro hodnoty pole IP protocol.

Postupně jsou zobrazeny sloupcové histogramy pro všechna zkoumaná pole záhlaví paketu. Obrázek 8.10 zobrazuje histogramy pro nejčastější protokoly vrstvy následující po IP záhlaví. Rozložení paketů do kategorií dle protokolu TCP odpovídá rozložení veškerému provozu. U protokolu UDP je vidět výrazné zastoupení kategorie L_4, neboť většina UDP toků obsahuje toky s malým počtem paketů. Podobné výsledky vykazuje i protokol ICMP. Ostatní protokoly jsou zastoupeny pouze v nepatrné míře a nemají dopad na odhad velikostí toků.

Z pohledu TCP toků jsou opět nejdůležitějším ukazatelem vývoje toku TCP příznaky. Z obrázku 8.11 je vidět, že existují kombinace nastavených bitů, které dokáží velmi přesně napovědět, do jaké kategorie může daný tok patřit. Většina paketů s nastaveným příznakem FIN patří do kategorie L_4. Stejně tak mají podobné rozložení příznaky RST + FIN, PSH + RST, ACK + RST + FIN. Příznak RST bývá nastaven u spojení, u kterých se nepodařilo dokončit navázání spojení a je potřeba je násilně ukončit. Osamocený příznak FIN značí, že paket s tímto příznakem nepotvrzuje přijetí přechozího paketu. To znamená, že je velmi pravděpodobné, že se v toku nepřenášela žádná data. Značně posunuté rozložení směrem ke kategorii L_4 mají rovněž pakety s příznakem SYN, ACK + FIN, ACK + RST. Paket s příznakem SYN je prvním paketem toku, který navazuje TCP spojení. Převážné zastoupení SYN pakety v kategorii L_4 je způsobeno neúspěšně navázanými spojeními a dále vysokým počtem toků v kategorii L_4. Osamocený příznak SYN se vyskytuje vždy jen



Obrázek 8.11: Sloupcové histogramy rozložení příslušnosti paketů do kategorií pro TCP příznaky.



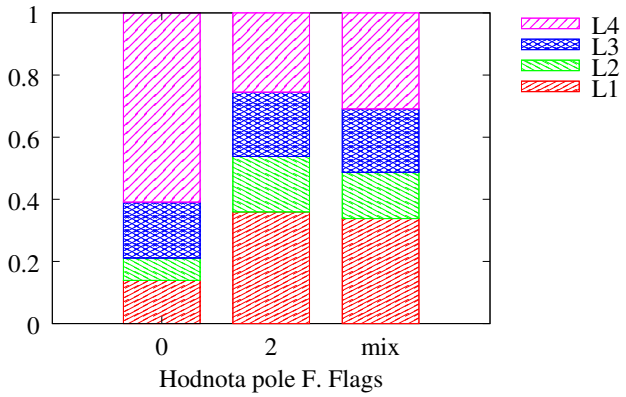
Obrázek 8.12: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků pro zvyšující se hodnotu TCP Window pole.

jednou pro daný tok. Rozložení těchto paketů do kategorií tedy odpovídá rozložení toků do kategorií. Rovněž kombinace příznaků ACK + FIN, ACK + SYN, kterými protistrana odpovídá na specifické události, jsou značně ovlivněny jejich výskytem na začátku a konci toku a tedy rozložení toků do kategorií. Běžnému rozložení se nejvíce blíží pakety s příznakem ACK a ACK + PSH. Důvodem je jejich nejčastější zastoupení v celkovém provozu.

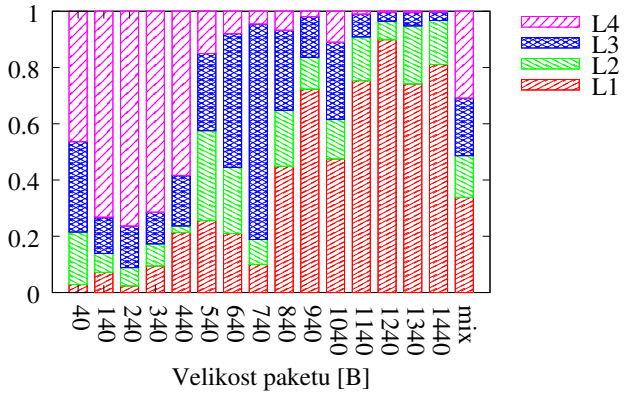
Pro analýzu pole TCP window jsou hodnoty sdruženy do několika skupin, každá o rozsahu 10000. V obrázku 8.12 je každá skupina reprezentována svou počáteční hodnotou. Navíc jsou přidány dvě speciální skupiny obsahující pouze jednu hodnotu a to hodnotu 0 a 65535. Je patrné, že pouze hodnota nula svým speciálním významem zvyšuje významně pravděpodobnost příslušnosti paketů do kategorie L_4. Dále lze pozorovat, že skupina hodnot 20001 obsahuje převážně pakety spadající do středních skupin toků L_2 a L_3. Ostatní hodnoty svým rozložením neliší od rozložení pro veškerý provoz.

Pole Fragment Flags je nastaveno na hodnotu 2 u více jak 85% paketů. Z toho plyne i velmi podobné rozložení paketů s touto hodnotou s rozložením všech paketů. Hodnota nula zvyšuje pravděpodobnost výskytu paketu náležícího toku z kategorie L_4. Naopak pravděpodobnost výskytu paketu kategorie L_1 je nižší.

Délka paketu je velmi dobrým ukazatelem z pohledu velikosti toku. Pro účely analýzy jsou délky sdruženy do skupin s rozsahem 100 B. Každá skupina je označena svou nejnižší



Obrázek 8.13: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků pro dvě nastavení pole Fragment Flags.

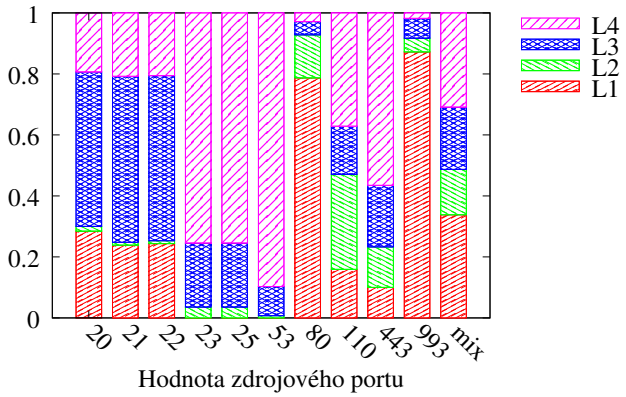


Obrázek 8.14: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků v závislosti na jejich velikosti paketů.

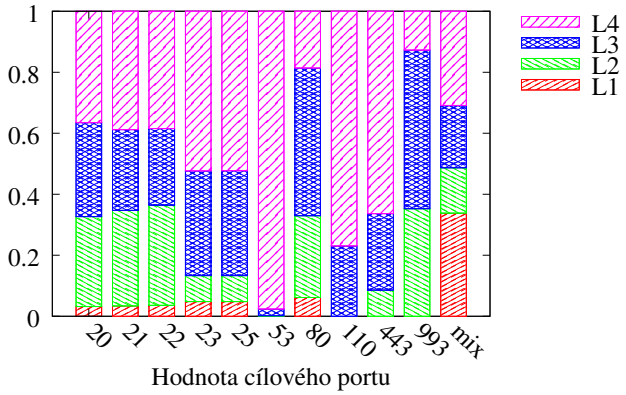
hodnotou. Obrázek 8.14 zobrazuje sloupcové histogramy pro všechny skupiny. U skupin obsahující krátké pakety je jasná převaha toků náležících do kategorie L_4. Naopak většina dlouhých paketů náleží do skupiny L_1, skupina L_4 nemá téměř žádné zastoupení.

Lze předpokládat, že na velikost toku má vliv aplikace, která tok využívá pro přenos dat. Některé aplikace přenáší typicky více dat, jiné méně. V analýze je pozornost zaměřena na dobře známé aplikace přenášející největší počet paketů na portech do hodnoty 1024. Nicméně pro rozpoznání příslušnosti paketu do kategorií může být rozpoznání provedeno i na základě portů z vyššího rozsahu.

Obrázky 8.15, 8.16 zobrazují rozložení příslušnosti paketů do kategorií pro vybraná čísla zdrojových, respektive cílových portů. Z obrázku 8.15 je dobře vidět, že většina paketů na portu 80 (HTTP) a 993 (IMAP) náleží do kategorie toků L_1. Přes port 80 je přeneseno největší množství paketů (přes 30%). Při porovnání obou obrázku je patrné, že pokud je vyhrazený port uveden jako zdrojový, pak jsou příslušné toky větší. Pokud je vyhrazený port uveden jako cílový, náleží pakety spíše do středních kategorií L_2, L_3. Toto chování lze očekávat vzhledem k modelu klient-server. Klient přistupuje k serveru (na vyhrazený cílový port) a žádá data (žádost má malou velikost). Data ze serveru jsou odeslána v toku, jehož zdrojový port odpovídá portu, na kterém server naslouchá. Tokem od klienta k serveru pak přichází pouze potvrzení o doručení paketů s daty, zatímco směrem od serveru ke klientovi



Obrázek 8.15: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků pro vybrané zdrojové porty.



Obrázek 8.16: Sloupcové histogramy rozložení příslušnosti paketů do kategorie toků pro vybrané cílové porty.

tečou samotná data. Tok ze serveru je tak významně větší.

8.3 Návrh klasifikátoru

Z předcházejících analýzy je patrné, že předpověď následující vzdálenosti paketu nebo velikosti toku na základě jednoho z polí v záhlaví paketu není příliš spolehlivá. Cílem je ale posílit správu cache o předpověď, která umožní zpřesnit heuristiku založenou na modelování historie příchodů paketů. Aby tato předpověď přinesla relevantní informaci, je nutné ji založit na vhodné kombinaci takových polí, která mají alespoň částečnou vypovídající hodnotu. Tvorba kombinované předpovědi je ekvivalentní konstrukci a učení klasifikátoru, který na základě příznaků (hodnot polí) určuje, zda se ve zkoumaných datech nachází určité objekty (v našem případě různé velikosti vzdáleností následujících paketů nebo velikosti toků).

Výběr vhodného klasifikátoru je ovlivněn několika kritérii, nejvíce se však na jeho výběru podílí vysoká přesnost a nízká výpočetní náročnost při vyhodnocení. Nízká výpočetní náročnost vyhodnocení je velmi důležitá, protože klasifikátor bude vyhodnocen s příchodem každého paketu. Zároveň je nutné, aby dosahoval dobrých výsledků klasifikace a byl tak platnou součástí rozhodování při správě cache. Na druhou stranu trénování klasifikátoru

může být sofistikovaný proces, který může být výpočetně a paměťově náročný, neboť trénování probíhá na zachycených a ohodnocených datech (každému paketu je přiřazena vzdálenost k následujícímu paketu nebo velikost příslušného toku). Na základě těchto kritérií je zvolena klasifikace pomocí rozhodovacího stromu. Rozhodovací strom rekurzivně dělí data na základě hodnot atributů (polí v záhlaví paketu) do menších podmnožin, dokud v podmnožině nezůstanou pouze instance (pakety) patřící jedné třídě (rozsahu vzdálenosti) nebo velikost podmnožiny neklesne pod stanovenou mez.

Rekuzivní algoritmus budování rozhodovacího stromu je dobře popsán v [52]. Zde je uveden stručný popis:

1. Nechť T je trénovací sada reprezentována kořenovým uzlem T.
2. Pokud všechny objekty v T naleží jedné třídě nebo třída obsahuje méně objektů než stanovuje mez, pak vytvoř uzel ohodnocený názvem třídy, jejíž prvky převažují v T a ukonči se.
3. Jinak zvol atribut B s prahovými hodnotami $b_1, b_2, \ldots, b_N$. Rozděl T do $T_1, T_2, \ldots, T_N$ dle $b_1, b_2, \ldots, b_N$.
4. Vytvoř strom s kořenovým uzlem pojmenovaným T' a uzly potomků $T_1, T_2, \ldots, T_N$ a hranami ohodnocenými hodnotami $b_1, b_2, \ldots, b_N$.
5. Vytvořeným stromem T' nahraď uzel T.
6. Rekurzivně aplikuj proceduru od bodu 2 pro uzly $T_1, T_2, \ldots, T_N$.

Kritickou součástí budování rozhodovacího stromu je volba atributu B a prahových hodnot $b_1, b_2, \ldots, b_N$ pro dělení množiny T. Volba parametru a určení prahových hodnot je provedena s ohledem na minimalizaci množství informace $I_B(T)$ zbývající v podmnožinách po rozdělení množiny T atributem B. Jinak řečeno, cílem je maximalizovat informační zisk G(B), který se vypočítá jako rozdíl entropie množiny T a součet entropie N podmnožin po rozdělení T podle B:

$$G(B) = I(T) - I_B(T),$$

$$I(T) = -\sum_{i=1}^{M} p_i \log_2 p_i,$$

$$I_B(T) = -\sum_{i=j}^{N} \frac{|T_j|}{|T|} I(T_j),$$

kde M je počet tříd, p_i je pravděpodobnost výskytu objektu i. Pro natrénování klasifikátoru je využito nástroje Weka [71] s implementací rozhodovacího stromu C4.5 [52].

Na základě předchozí analýzy jsou jako vstup klasifikátoru následujících vzdáleností vybrána pole IP protokol, TCP Flags, TCP Window a Total Length. Počet tříd a jejich rozsahy byly určeny tak, aby poskytovaly dostatečné rozlišení pro potřeby správy cache. Pro účely klasifikace jsou vzdálenosti sdruženy do čtyř tříd dle jejich délky *krátké* ... $\langle 0, 1000)$, *střední* ... $\langle 1000, 100000)$, *dlouhé* ... $\langle 100000, 1000000)$ a *nekonečno* ... $\langle 1000000, \infty)$. První dvě třídy reflektují pozitivní předpověď, kdy je výhodné stav toku uchovat v cache, zbylé dvě třídy představují negativní předpověď. Rozdělení do více tříd by znamenalo ztížení tvorby klasifikátoru, jednotlivé třídy by měly menší zastoupení v trénovací sadě a zvýšilo

Tabulka 8.3: Matice zmatení klasifikátoru K_{real} (confusion matrix).

	krátké	střední	dlouhé	nekonečno
krátké	94,9	4,5	0,0	0,7
střední	42,6	55,5	0,1	1,8
dlouhé	34,6	59,1	1,1	5,2
nekonečno	21,1	28,2	0,0	50,6

Tabulka 8.4: Rozložení vzdáleností do tříd.

Třídy	krátké	střední	dlouhé	nekonečno
Zastoupení [%]	65,9	27,3	1,4	5,4

by se riziko špatného výsledku klasifikátoru. Navíc správa cache nevyžaduje přesnou hodnotu nadcházející vzdáleností mezi pakety, která je i v běžném provozu zatížena šumem. Rozšířené datové sady, které obsahují vzdálenosti následujících paketů v toku, jsou upraveny a na místo samotného údaje o vzdálenosti je uveden název třídy. Následně jsou datové sady rozděleny na dvě nepřekrývající se části. Jedna třetina datové sady je využita pro trénování a zbylé dvě třetiny pro ohodnocení přesnosti natrénovaného klasifikátoru. Výstupem klasifikátoru je název třídy, do které náleží nadcházející vzdálenost. Samotný klasifikátor je pojmenován jako K_{real}.

Nastavení parametrů trénování klasifikátoru je popsáno v příloze 11.4 včetně natrénovaného klasifikátoru v podobě stromu. Výsledky ohodnocení klasifikátoru na části sady *Mawi-2010/04/14-14:00* ukazují, že natrénovaný klasifikátor je schopen u 80% paketů správně predikovat vzdálenost následujícího paketu toku. Detailní rozbor výsledků je uveden v tabulce 8.3. Tato tabulka je tvořena tzv. maticí zmatení klasifikátoru. Tato matice ve svých řádcích uvádí procentuální rozložení množství paketů jedné třídy do všech tříd. Tedy udává, jak ohodnotil klasifikátor pakety dané třídy. Na diagonále matice se nachází správně ohodnocené pakety dané třídy a na zbývajících pozicích jsou uvedeny špatně klasifikované pakety. V ideálním případě by 100%paketů třídy X bylo přiřazeno opět třídě X a ostatním třídám v řádku by bylo přiřazeno 0% paketů dané třídy.

Z tabulky je patrné, že nejlépe jsou predikovány krátké vzdálenosti paketů, kdy se daří úspěšně predikovat více než 94% vzdáleností spadajících do této třídy. Střední vzdálenosti jsou ve 42% případů predikovány klasifikátorem jako krátké a z 55% správně jako střední. Záměna části středních vzdáleností za krátké by nemusela mít negativní vliv na rozhodování při správě cache, neboť příslušné toky obsahující obě tyto vzdálenosti se vyplatí udržovat v cache. Nejhorší výsledky dosahuje klasifikátor při predikci dlouhých vzdáleností, kdy téměř dvě třetiny dlouhých vzdáleností jsou predikovány jako střední a třetina jako krátké. Tento stav by opět nemusel hrát velkou roli ve výsledné správě cache a to ze dvou důvodů. Počet těchto vzdáleností je malý. Dle tabulky 8.4 zobrazující rozložení hodnot vzdáleností připadá na celkové množství vzdáleností pouze 1,4% dlouhých. Dále jsou to opět vzdálenosti, které by mohly být na hranici, zda se daný tok vyplatí nebo nevyplatí v cache udržovat. Poslední třída *nekonečno* je správně klasifikována z jedné poloviny. Druhá polovina je klasifikována jako střední nebo krátká. Zde je poměrně velké riziko, že klasifikátor špatně napoví správě cache třídu vzdálenosti následujícího paketu. Nicméně v případě dobré nápovědy významně ušetří místo v cache. Správy vyvinuté bez nápovědy vkládají nové toky do první

Tabulka 8.5: Matice zmatení klasifikátoru $K_{V,real}$ velkých toků.

	L_1	L_2	L_3	L_4
L_1	98,9	0,8	0,1	0,1
L_2	8,7	88,3	1,6	1,4
L_3	1,5	1,7	87,6	9,2
L_4	0,2	0,4	3,9	95,5

třetiny spravovaného řádku. V takovém případě trvá poměrně dlouho, než se tok obsahující pouze jeden paket dostane aktivitou ostatních toků na konec řádku, kde může být odstraněn z cache. Pokud by správa cache na základě nápovědy vložila takový tok například do třetí třetiny řádku, pak by významně pomohla šetřit místo v cache pro jiné aktivní toky. Pokud je nekonečná vzdálenost chybně klasifikována jako krátká, pak může nastat situace, kdy tok zůstane v cache ještě déle než při původní správě cache. Je tedy na správě cache, kterou dodatečnou informaci a jak ji bude využívat, aby bylo dosaženo zlepšení.

Pro trénování klasifikátoru velikosti toků $K_{V,real}$ je každý paket datové sady označen značkou, do které kategorie náleží. Jedna třetina datové sady je využita pro trénování a zbylé dvě třetiny pro ohodnocení. Pro klasifikaci je opět využito klasifikátoru pomocí rozhodovacího stromu s implementací C4.5 v nástroji Weka [71]. Výsledky ohodnocení klasifikátoru na části sady *Mawi-2010/04/14-14:00* ukazují, že natrénovaný klasifikátor je schopen u 94% paketů správně predikovat příslušnost daného toku do kategorie. Detailní rozbor výsledků je uveden v tabulce 8.5. Matice zmatení klasifikátoru ve svých řádcích uvádí procentuální rozložení množství paketů jedné kategorie do všech tříd při klasifikaci klasifikátorem. Tedy udává, jak přesně ohodnotil klasifikátor pakety dané třídy.

Z tabulky je patrné, že nejlépe je predikována kategorie L_1, kdy se daří úspěšně predikovat téměř 99% paketů spadajících do této kategorie. Kategorie L_2 a L_3 jsou u přibližně 12% případů predikovány odlišně od své skutečné kateogorie. Kategorie L_4 je opět úspěšně predikována v 95% případů, s největší pravděpodobností 3,9% chybné predikce jako sousední L_3.

8.4 Kombinace výstupu klasifikátoru a správy cache

Klasifikátor lze zřejmě využít přímo pro rozhodování o odstranění stavu toku z cache na místo běžné správy cache. V takovém případě je výsledek klasifikátoru uložen do stavu toku. V době potřeby uvolnění místa je vyhledán stav toku s klasifikací *nekonečno* (v případě, že není žádný stav klasifikován do třídy nekonečno, pak je vyhledáván stav s další nejvyšší možnou třídou). Rozhodování čistě na základě výsledku reálného klasifikátoru K_{real} neposkytuje ale dobré výsledky. Experimenty ukazují, že taková správa dosahuje v případě konfigurace cache pro 8192 toků a využití datové sady *Mawi-2010/04/14-14:00* 20% pravděpodobnosti výpadků. Naproti tomu správa cache GARP bez klasifikátoru dosahuje méně než 6% pravděpodobnosti výpadků.

Z tohoto důvodu je vhodné výstup klasifikátoru využít spíše jako dodatečnou informaci pro zlepšení rozhodování správy cache. V ideálním případě by výstupy klasifikátoru měly podávat chybějící informaci k informaci, kterou si správa cache uchovává sledováním předchozích přístupů k toku. Například pokud správa cache udržuje určitý tok na konci řádku, kde hrozí jeho odstranění, ale výstup klasifikátoru predikuje správně následující

přístup v blízké budoucnosti, pak výstup klasifikátoru doplnil chybějící informaci. Proto je třeba vyvinout správu cache pomocí GA dohromady s nápovědou od klasifikátoru. V takovém případě může GA upravit využití samotné nápovědy a zároveň i samotnou správu cache tak, aby byl umocněn přínos klasifikátoru. Například v teoretické situaci, kdy klasifikátor a původní správa cache jsou v naprosté shodě, může GA vyvinout zcela odlišnou správu cache, která bude zaměřena na uchování informace, kterou neposkytuje klasifikátor.

Přímočarým způsobem řešení kombinace výstupu klasifikátoru a správy cache $S = (s, V)$ je vytvoření samostatných správ pro každou třídu výsledku $S_{trida} = (s_{trida}, V_{trida})$. Výsledná správa S tak obsahuje čtyři správy $S = \{S_{kratka}, S_{stredni}, S_{dlouha}, S_{nekonecno}\}$. Na základě výstupu klasifikátoru je využita příslušná správa cache. Toto řešení má ovšem velkou nevýhodu ve značném zvětšení prohledávaného prostoru. Pokud původní prohledávaný prostor obsahoval $1,310^{15}$ možných řešení, pak spojený prostor čtyř správ pamětí, které nelze vyvíjet nezávisle na sobě, obsahuje $(1,3.10^{15})^4 = 2,5.10^{60}$ řešení.

Vzhledem k tomuto nárůstu stavového prostoru je navržen jiný způsob kombinace výstupu klasifikátoru se správou cache. Vzájemná kombinace je vybudována na úpravě pozice, na kterou by správa cache stav toku v řádku umístila. Jinými slovy, poté co správa cache vrátí novou pozici stavu, je tato pozice upravena na základě výstupu klasifikátoru. Aby bylo možné ovlivňovat míru úpravy pozice pro každou třídu, je zaveden vektor úprav $U = (u_{kratka}, u_{stredni}, u_{dlouha}, u_{nekonecno})$. Nová pozice stavu toku f v čase $t+1$ se vypočítá jako:

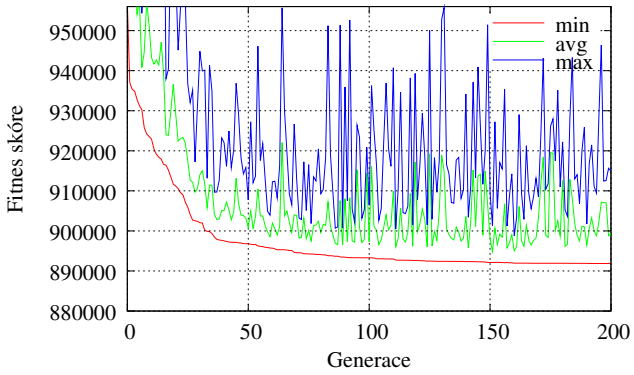
$$pos_{t+1}(f) = v_{pos_t(f)} + u_{trida}, \tag{8.1}$$

kde dolní index *trida* je výsledek klasifikace příchozího paketu a u_{trida} je skalár nabývající hodnot $-16 \ldots 17$. Rozšířená správa cache je pak definována jako $S_{ext} = (s, V, U)$ a pro čtyři třídy se velikost prostoru řešení pohybuje okolo 1.10^{21}. GA vyvíjí vektor U společně s ostatními složkami správy. Pro vývoj složek V a s je využito stávajících dobře fungujících operátorů překódování a kontrolované mutace. Na základě experimentů je nejvhodnějším operátorem pro změnu vektoru U zvolena běžná mutace, využití křížení nepřináší žádné zlepšení.

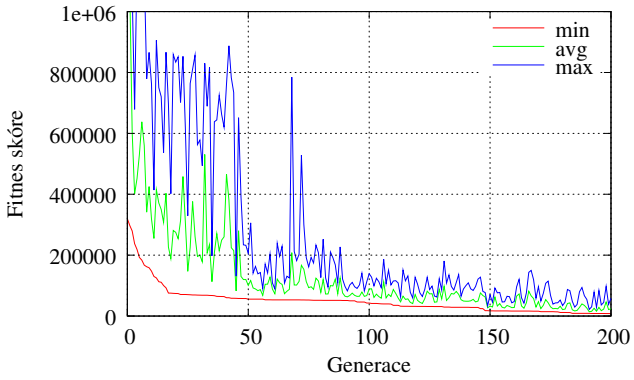
Pro vývoj rozšířené správy cache GARP_K s natrénovaným klasifikátorem (reálný natrénovaný klasifikátor jako K_{real}) bylo spuštěno 10 běhů GA. Průběh ohodnocení populace (průměr minim, maxim a průměru ze všech běhů) je zobrazen na obrázku 8.17.

Nalezená rozšířená správa cache GARP_K se výrazně liší od původní správy cache GARP bez nápovědy klasifikátoru. Srovnání jejich klasifikačních vektorů U, aktualizačních vektorů V a míst vkládání s je uvedeno v příloze v tabulce 11.4.

Shodně s přístupem zvoleným pro kombinaci GARP a K_{real} je zkombinován i klasifikátor $\text{K}_{V,real}$ se správou GARP-V2. Kombinace je provedena pomocí úpravy pozice při přesunu nebo vložení toku podle výsledku klasifikátoru shodně se způsobem kombinace odhadu následujících vzdáleností (viz rovnice (8.1)). Vzniká tak rozšířená správa paměti $\text{GARP-V2}_{K_V}-\text{K}_{V,real}$. Pro vývoj rozšířené správy cache $\text{GARP-V2}_{K_V}-\text{K}_{V,real}$ s reálným natrénovaným klasifikátorem je spuštěno 10 běhů GA. Průběh ohodnocení populace (průměr minim, maxim a průměru ze všech běhů) je zobrazen na obrázku 8.18. Průběh ohodnocení odpovídá očekávanému průběhu hodnotící funkce. V prvotních generacích jsou vyvíjena různorodá řešení, která často selhávají. Následně jsou nalezena řešení, která úspěšně kombinují klasifikátor s vektorem V správy paměti. Tato řešení jsou v dalších generacích optimalizována a konvergují k velmi dobrému výsledku. Nalezené $\text{GARP-V2}_{K_V}-\text{K}_{V,real}$ nalezené pro různé datové sady jsou zachyceny v tabulce 11.7 v příloze.



Obrázek 8.17: Průběh ohodnocení populace (průměr z deseti běhů) při vývoji správy $\text{GARP}_K - \text{K}_{real}$.



Obrázek 8.18: Ukázka průběh ohodnocení populace (průměr z deseti běhů) při vývoji správy $\text{GARP-V2}_{K_V} - \text{K}_{V,real}$ na datové sadě *Mawi-14:15*.

8.5 Výsledky

Je provedeno několik experimentů za účelem prozkoumání vlastností klasifikátoru a vyvíjené správy cache. Výsledky těchto experimentů shrnuje tabulka 8.6. Nejprve je provedeno měření, jehož účelem je odhalení úspěšnosti ideálního klasifikátoru (ideální klasifikátor má 100% přesnost předpovědi) bez využití další správy cache. V tabulce je označen výsledek tohoto experimentu jako K_{ideal}. Ve své podstatě se jedná o správu cache pomocí FITF s tím rozdílem, že klasifikace poskytuje pouze čtyři úrovně rozlišení vzdálenosti následujícího paketu a nemůže být tedy tak přesná jako FITF. V případě experimentu s *Mawi-14:15* se pravděpodobnost výpadků přiblížila na 1,5% k FITF. Skutečný natrénovaný klasifikátor značený jako K_{real} dosahuje při samostatném použití mnohem horších výsledků, jak bylo uvedeno výše, přibližně $p_\nu = 20\%$. Následně je uveden výsledek úspěšnosti správy GARP bez klasifikátoru, tedy správy cache vyvinuté v předchozích kapitolách.

Vývoj správy s klasifikátorem GARP_K proběhl ve dvou variantách. Nejprve byla vyvinuta správa cache kombinující $\text{GARP}_K - \text{K}_{ideal}$. Z výsledků v tabulce je patrné, že tato správa předčila svými výsledky samostatnou K_{ideal} a nejvíce se přiblížila výsledkům teoretické FITF. Je vidět, že GA je schopen využít výsledky klasifikátoru a přizpůsobit GARP_K danému klasifikátoru tak, aby bylo dosaženo zlepšení oproti samostatnému K_{ideal}. Výsledná správa se svou úspěšností přiblížila na 1% od FITF. Dále proběhl vývoj GARP_K s reálným natrénovaným klasifikátorem K_{real}. GA mohl upravit správu cache tak, aby

Tabulka 8.6: Porovnání úspěšnosti správ cache s klasifikátorem.

	Mawi-14:15		*Snjc-13:05*		*Vut-15:05*	
Správa	M_1	M_2	M_1	M_2	M_1	M_2
K_{real}	22,6	288,5	34,1	473,5	10,2	354,4
GARP	5,3	144,8	4,1	144,1	2	151,6
$GARP_K - K_{real}$	4,4	139,8	3,8	139,2	1,9	150
K_{ideal}	3,4	127,9	3,2	134,5	1,2	128,2
$GARP_K - K_{ideal}$	2,7	122,9	2,4	127,8	1	126
FITF	1,9	115,6	0,9	109,3	0,9	122,7

nepřesné výsledky klasifikátoru byly eliminovány. $GARP_K$ s reálným klasifikátorem dosahuje lepších výsledků i přes nepřesnosti klasifikátoru než GARP bez klasifikátoru. Dle očekávání dosahuje $GARP_K$ s K_{real} horších výsledků než $GARP_K$ s K_{ideal}. Zlepšení oproti nejlepší reálné správě cache GARP je o jedno procento p_ν a o sedm procent v počtu nesprávně odstraněných stavů toků na datové sadě *Mawi-14:15*. Uvážíme-li, že GARP dosahuje nízké hodnoty $p_\nu = 5{,}3\%$, která je pouze 3,4% od dosažitelného optima, pak zlepšení o jedno procento lze považovat za významný přínos umocněný špatnými výsledky sofistikovanějších správ cache.

Pro zajímavost jsou provedeny experimenty ukazující schopnost přizpůsobit $GARP_K$ danému klasifikátoru. Pokud jsou zaměněny klasifikátory u obou $GARP_K$ správ, tj. $GARP_K$ vyvíjená s K_{ideal} je ohodnocena při používání K_{real} a naopak $GARP_K$ vyvíjená s K_{real} je ohodnocena při využívání K_{ideal}, pak jsou výsledky úspěšnosti vyhledání horší než u správ (v závislosti na datové sadě se liší v řádu desetin procent), které jsou vyvíjeny a následně ohodnoceny se stejnými klasifikátory. Z vektorů těchto správ uvedených v příloze tabulce 11.4 je vidět, že složky $GARP_K$ (s, V, U) se pro oba klasifikátory významně liší.

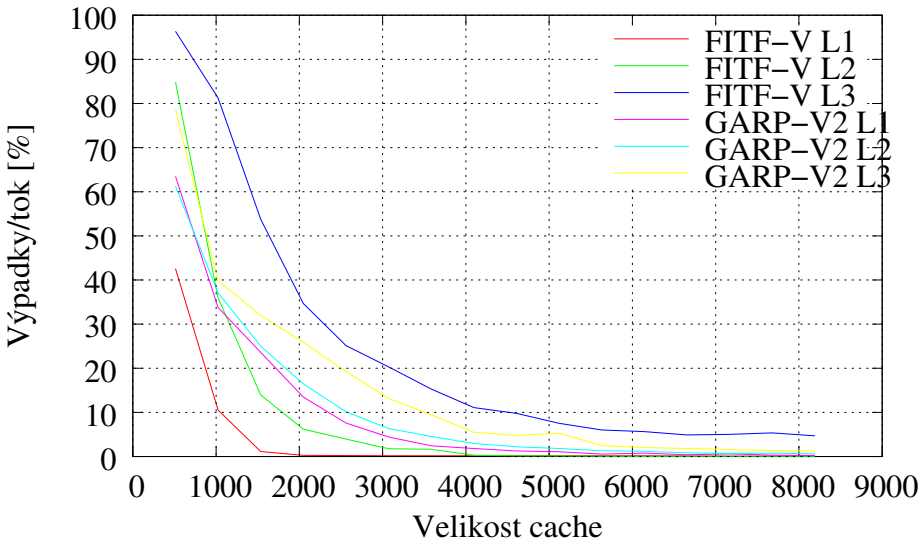
Porovnání výsledků $GARP\text{-}V2_{K_V} - K_{V,real}$ s FITF-V a $K_{V,ideal}$ je uvedeno v tabulce 8.7. Ideální klasifikátor $K_{V,ideal}$ je schopen velmi dobře spravovat cache velkých toků. Na rozdíl od FITF-V má informaci pouze o velikosti toku, nikoliv však o příchodech následujících paketů. Výsledky ideálního klasifikátoru se blíží ideální správě. Tento výsledek lze očekávat, neboť je způsoben tím, že v převážné většině případů postačí odstranit tok z kategorie L_4 pro uvolnění místa v cache. Informace o následujícím příchodu paketu v toku je důležitá pouze v případě, ve kterém je nutné se rozhodnout mezi toky stejné kategorie L_3, L_2, L_1.

Správa $GARP\text{-}V2_{K_V} - K_{V,real}$ dosahuje téměř shodných výsledků jako správa FITF-V. Odhad velikosti toku na základě příznaků ze záhlaví paketu pomáhá $GARP\text{-}V2_{K_V} - K_{V,real}$ zásadně snížit množství odstraněných stavů velkých toků z cache.

Dále je proveden experiment s cílem sledovat chování $GARP\text{-}V2_{K_V} - K_{V,real}$ v závislosti na velikosti cache. FITF-V preferuje se zmenšující se cache velké toky, kdy při velikosti cache větší jak 2048 položek nedochází k výpadkům stavů toků kategorie L_1. Shodně prefereje FITF-V toky kategorie L_2. Proto je pro FITF-V počet výpadků na tok kategorie L_3 vyšší než u $GARP\text{-}V2_{K_V} - K_{V,real}$. $GARP\text{-}V2_{K_V} - K_{V,real}$ není schopna obětovat stavy patřící tokům z kategorie L_3 nebo L_2 ve prospěch kategorie L_1 do stejné míry jako FITF-V. Nápověda v podobě klasifikátoru dovoluje $GARP\text{-}V2_{K_V} - K_{V,real}$ dosahovat podobných výsledků jako FITF-V pro velikost cache 4096 stavů a vyšší.

Tabulka 8.7: Porovnání úspěšnosti správy cache velkých toků.

Mawi-14:15	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
$K_{V,real}$	10%	31%	48%	30%	97%	160%
GARP-V2	4%	8%	12%	9%	21%	34%
GARP-V2$_{K_V}$ $-K_{V,real}$	2%	3%	8%	6%	6%	22%
$K_{V,ideal}$	0%	1%	21%	0%	7%	63%
FITF-V	0%	1%	6%	1%	2%	36%
Snjc-13:05	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
$K_{V,real}$	15%	68%	95%	30%	159%	183%
GARP-V2	0%	7%	9%	0%	18%	17%
GARP-V2$_{K_V}$ $-K_{V,real}$	0%	3%	4%	0%	8%	7%
$K_{V,ideal}$	0 %	0%	1%	0%	0%	2%
FITF-V	0%	0%	0%	0%	0%	0%
Vut-15:05	M_3			M_4		
Správa	L_1	L_2	L_3	L_1	L_2	L_3
$K_{V,real}$	41%	65%	74%	170%	505%	523%
GARP-V2	7%	11%	20%	5%	7%	19%
GARP-V2$_{K_V}$ $-K_{V,real}$	3%	4%	12%	4%	7%	18%
$K_{V,ideal}$	2%	3%	10%	3%	6%	24%
FITF-V	2%	3%	8%	3%	6%	19%



Obrázek 8.19: Procentuální poměr výpadků vůči počtu toků pro různé velikosti cache spravovanou GARP-V2$_{K_V}$ $-K_{V,real}$ (ohodnoceno na *Mawi-14:15*).

Kapitola 9

Systém pro organizaci síťového provozu

Množství síťového provozu na vysokorychlostních linkách vyžaduje urychlení jeho zpracování na specializovaných výpočetních prostředcích. Mezi používané prostředky urychlení jsou používány například programovatelná hradlová pole (FPGA – Field Programmable Gate Arrays), síťové procesory nebo GPU.

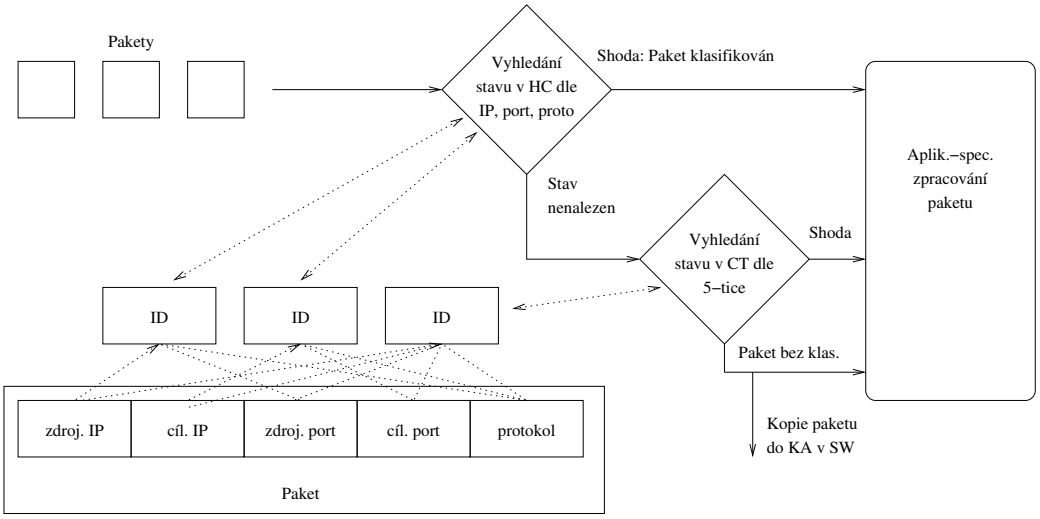
Pozornost je zaměřena na urychlení stavového zpracování síťového provozu s využitím FPGA. K urychlení jsou využity síťové karty NetFPGA vybavené programovatelným hradlovým polem s technologií Virtex-II-Pro od firmy Xilinx. Tyto síťové karty jsou propojeny s hostitelským počítačem přes sběrnici PCI-Express, resp. PCI.

Pro tento systém je navržena a implementována úloha třídění a tvarování síťového provozu (traffic-shaping) nazvaný AtoZ. Tato úloha je rozdělena mezi síťovou kartu a hostitelský počítač tak, aby přenos dat mezi počítačem a kartou byl minimalizován. Vznikají dvě úrovně zpracování, datová na síťové kartě a kontrolní v hostitelském počítači. Díky rozdělení je možné na kartě zpracovat většinu provozu, aniž by musel být přenášen mezi kartou a hostitelským počítačem.

9.1 Motivace a výzvy

Internetová síť využívá *best-effort* model při doručování paketů. Pokud je možné paket bez komplikací doručit, pak je doručen. Pokud dojde při přenosu k neočekávaným událostem (například zahlcení síťového prvku), pak je paket bez ohledu na jeho obsah zahozen. Služby, které jsou citlivé na dostupnou šířku pásma, se stanou při zahlcení nedostupné. Cílem organizace a tvarování provozu je omezit přenosovou šířku pásma přidělovanou jednotlivým aplikacím nebo uživatelům. Citlivým a kritickým aplikacím je garantována šířka pásma, zatímco další mohou být omezeny nebo blokovány.

V souvislosti s úlohou tvarování a organizace síťového provozu jsou kritické dvě operace, rozpoznání aplikace v toku a udržení stavové informace po celou dobu aktivity toku. Rozpoznání aplikace pouze na základě vyhrazených portů má několik překážek. Značná část provozu využívá porty mimo vyhrazený rozsah, v šifrovaném spojení nemusí být hodnota portů dostupná a některé aplikace mohou využívat vyhrazené porty, přestože jim nebyly přiřazeny. Z tohoto důvodu se využívají alternativní přístupy založené na vyhledávání řetězců v obsahu toku nebo na klasifikaci pomocí hodnot v záhlaví prvních několika paketů. Průběh klasifikace je uložen do stavu toku. Po určité době obsahuje většina stavů typ roz-



Obrázek 9.1: Schéma hierarchického klasifikačního systému.

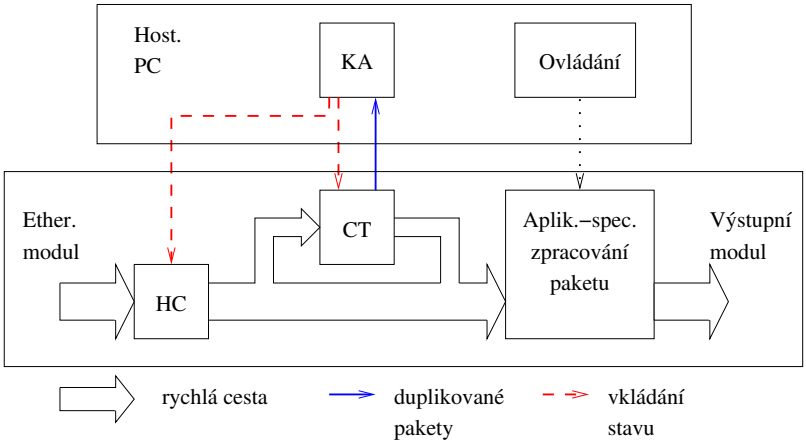
poznané aplikace. Pokud dojde ke ztrátě stavu, pak rozpoznání a správné zpracování je ve většině případů nemožné.

Systém AtoZ řeší rozpoznání aplikace pomocí klasifikátoru paketů. Dále využívá dvě cache toků s různou definicí toku. Správa těchto cache je zaměřena na velké toky. Úloha celého systému je rozdělena mezi hardware a software. FPGA dovoluje dosáhnout zpracování paketů bez ztrát na rychlosti linky a zároveň představuje pouze nízké zpoždění v cestě paketu. V hostitelském počítači běží komplexní program rozpoznávající aplikace v tocích.

9.2 Návrh systému

AtoZ je navrženo jako hierarchický systém tří klasifikátorů. První dvě úrovně jsou tvořeny dvěmi cache. Tyto cache přebírají výsledky od třetího klasifikátoru a nejsou tak klasifikátory v pravém slova smyslu. Třetí klasifikátor (označený KA - klasifikátor aplikací) je natrénovaný rozhodovací strom rozpoznávající aplikace v tocích na základě vlastností toku v jeho počáteční fázi. Schéma hierarchie klasifikátorů je zobrazeno na obrázku 9.1.

Cache první úrovně je nazvána Host Cache (HC). Tato cache definuje tok jako množinu paketů se stejnou trojicí cílové IP adresy, cílového portu a protokolu, resp. zdrojové IP adresy, zdrojového portu a protokolu. HC slouží pro uchování pravidel o zjištěných aktivních aplikacích, které běží na daném serveru a naslouchají na určitém portu. Paket splňující pravidlo v cache je označen značkou, podle níž se řídí jeho další zpracování. Pravidlo obsahuje IP adresu, port, protokol a dvě značky určující jaká akce se má provést, pokud se jedná o zdrojová nebo cílová pole. S příchodem každého paketu jsou provedena dvě vyhledání, jedno na základě cílových polí a druhé na základě zdrojových polí. Pokud paket odpovídá pravidlu v HC, je označen příslušnou značkou a přeposlán na další zpracování. Shoda paketu se dvěma pravidly by nastala v případě, že by server použil pro komunikaci s jiným serverem port, na kterém naslouchá jeho aplikace. K tomu nedochází a je velmi pravděpodobné, že by komunikace v takovém případě selhala. Pravidla do HC jsou vkládána dynamicky podle toho, jak KA objevuje běžící aplikace na serverech. Ve shodě s předchozími pozorováními [49, 50] je využito heavy-tail rozložení provozu mezi komunikující hosty. HC



Obrázek 9.2: Schéma systému AtoZ.

udržuje pravidla pro nejvíce vytížené servery a aplikace. Tím značně odlehčuje následujícím modulům.

HC není schopna označit správně pakety toku, pokud neobsahuje pravidlo se vztahem mezi aplikací a trojicí IP adresy, portu a protokolu daného toku. Aby bylo možné rychle a bez zpoždění zpracovat i zbývající neoznačenou část provozu, obsahuje hierarchie další cache. Tato cache je pojmenována CT a slouží pro značení toků definovaných na základě pětice zdrojové a cílové IP adresy, zdrojového a cílového portu, protokolu. Navíc CT umožňuje duplikovat prvních δ paketů neoznačeného toku a přeposlat tyto pakety do KA. KA rozpozná aplikaci přenášenou v toku a zaznačí tuto informaci zpět do stavu toku v CT. Pakety, které odpovídají existujícímu pravidlu-stavu toku, jsou označeny a přeposlány na zpracování. Pakety, které nejsou označeny (tedy i prvních δ paketů toku), obdrží vyhrazenou značku a jsou přeposlány na zpracování jako nerozpoznané.

KA přijímá prvních δ paketů neoznačeného toku. Rozpoznání aplikace probíhá pomocí rozhodovacího stromu. Rozhodování se provádí na základě vybraných polí ze záhlaví paketu. Rozhodovací strom je natrénován offline na malém vzorku dat. Tento vzorek byl předem analyzován a každý tok byl označen značkou určující aplikaci, jejíž data tok přenáší. Detaily trénování klasifikátoru pro tento systém jsou popsány v [41]. Celkem je využito 12 polí sbíraných z prvních pěti paketů toku (tj. $\delta = 5$). Toky jsou rozdělovány celkem do 13 tříd dle typu dat (třídy jsou WEB, MAIL, BULK, ATTACK, CHAT, P2P, DATABASE, MULTIMEDIA, VOIP, SERVICES, INTERACTIVE, GAMES, and GRID). To znamená, že pokud tok přenáší interaktivní textovou komunikaci programu MSN, pak je zařazen to třídy „CHAT“.

Pro experimenty s hierarchickým klasifikátorem je navržen triviální aplikačně-specifický modul pro zpracování označeného paketu. Na základě značky přesměruje provoz na zvolený port a změní cílovou MAC adresu.

9.3 Implementace

Schéma hierarchického klasifikátoru je rozděleno mezi FPGA a hostitelský počítač, jak je znázorněno na obrázku 9.2. KA s pomocnými moduly tvoří řídicí cestu, která řídí značení paketu v datové cestě. KA si na základě přeposlaných paketů vytváří stav ke každému toku. Tento stav obsahuje sbírané statistiky pro vstup klasifikátoru aplikací. Poté, co je

rozpoznána aplikace, je výsledek zapsán do CT. V případě, že více toků bylo pro daný server a port ropoznáno shodně, je výsledek zapsán do HC.

Moduly cache, HC a CT jsou i s modulem pro zpracování paketu umístěny do FPGA. Tyto moduly tvoří hlavní datovou cestu pro analýzu a zpracování paketů systémem. Moduly pracují na šířce slova 64 bitů na frekvenci 125 MHz a jsou schopny zpracovat až 8 milionů paketů za vteřinu. Tento výkon je dostatečný pro zpracování všech čtyř gigabitových síťových rozhraní nezávisle na velikosti paketu. Navržené moduly pracují v tzv. *cut-through* módu. To znamená, že zpracování paketu začíná již s příchodem prvních slov, a jakmile je po několika hodinových taktech k dispozici výsledek, je paket přeposílán k dalšímu zpracování bez toho, aby se čekalo na příjem celého paketu.

Po příjmu paketu síťovými buffery je poslán do HC. HC získá IP adresu, porty a číslo protokolu a vypočítá dvě hash hodnoty. První je vypočtena ze zdrojové IP adresy, zdrojového portu a protokolu a druhá z cílové IP adresy, cílového portu a protokolu. Tyto hash hodnoty jsou použity pro vyhledání stavu v HC. Popis vyhledání s pomocí hash hodnot je uveden v sekci 9.3.1. Pokud je vyhledání úspěšné, pak je paket označen třídou aplikace. Úspěšně označený paket projde FC beze změny a je zpracován ve výstupním modulu. V opačném případě CT získá pětici IP adres, portů a protokolu a vypočítá hash hodnotu, kterou použije pro vyhledání stavu ve své cache. Pokud je vyhledání úspěšné, obsahuje stav čítač paketů a případně i třídu aplikace, pokud již byl tok klasifikován. Čítač slouží pro sledování počtu paketů od začátku toku a zajišťuje, že do KA je zasláno pouze prvních δ paketů. Pokud je vyhledání neúspěšné, je založen stav nový a paket odchází neoznačen a zároveň je kopie přeposlána do KA. Označené i neoznačené pakety pokračují do aplikačně-specifického modulu pro zpracování paketů, kde je dle nahraných pravidel změněna cílová MAC adresa a výstupní port. Rozšířený modul by mohl implementovat prioritní fronty pro upřednostnění kritického provozu, dále filtry a omezovače provozu pro kontrolu nepovolených aplikací.

9.3.1 Vyhledání v cache

Vyhledání v obou modulech HC a CT je založeno na vyhledání v hash tabulce, která odpovídá N-cestné cache. Tento koncept je založen na využití hash hodnoty pro rozdělení cache na nezávislé řádky. Vyhledání probíhá následovně. Nejprve je spočítána hash z polí v záhlaví paketu dle definice toku pro danou cache. Hash hodnota odpovídá adrese řádku. Řádek je prohledán, zda obsahuje hledaný stav.

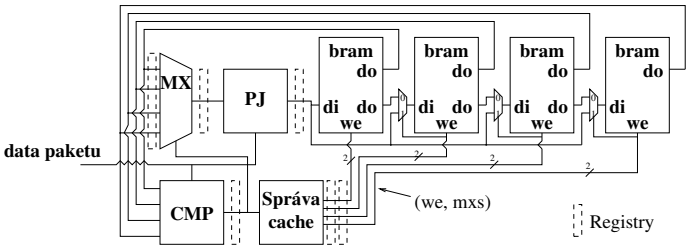
Protože paměťové zdroje v FPGA jsou omezené, je v každém stavu na místo uplného identifikátoru toku uložen pouze otisk v podobě další části hodnoty hash, která nebyla využita při vyhledání řádku. Tím se značně zvyšuje množství stavů, které jsou obě cache schopny uchovat. Zároveň se ale zvyšuje pravděpodobnost nesprávné identifikace toku, pokud dojde ke kolizi (dva různé identifikátory toků budou mít shodný otisk, respektive celou hash hodnotu). Pravděpodobnost kolize lze snížit na akceptovatelnou úroveň prodloužením otisku (počtem jeho bitů b). Pravděpodobnost kolize p_k odpovídá řešení tzv. narozeninového paradoxu. Pro 2^h řádků je délka hash hodnoty pro adresu řádku a pro otisk rovna $h + b$. Pravděpodobnost kolize p_k odpovídá

$$p_k = 1 - \frac{m!}{m^n(m-n)!} = 1 - \frac{2^{(h+b)}!}{2^{(h+b)^n}(2^{(h+b)}-n)!} \tag{9.1}$$

$$\approx 1 - e^{-\frac{(n-1)n}{2\times 2^{(h+b)}}}, \tag{9.2}$$

Počet stavů	Délka hash $(h+b)$		
	40	44	48
4K	$1.9 \cdot 10^{-6}$	$1.2 \cdot 10^{-7}$	$7.5 \cdot 10^{-9}$
8K	$4.1 \cdot 10^{-5}$	$3.2 \cdot 10^{-7}$	$2.4 \cdot 10^{-8}$
16K	$1.2 \cdot 10^{-4}$	$7.6 \cdot 10^{-6}$	$4.8 \cdot 10^{-7}$
32K	$4.9 \cdot 10^{-4}$	$3.1 \cdot 10^{-5}$	$1.9 \cdot 10^{-6}$

Tabulka 9.1: Pravděpodobnost kolize (p_k).



Obrázek 9.3: Architektura hashovací tabulky v FPGA.

kde m je počet možných hodnot hash funkce a n je počet uložených stavů v cache. Tabulka 9.1 poskytuje vyčíslení p_k pro několik konfigurací délky hash a počet stavů n.

9.3.2 Správa cache a implementace v FPGA

Obě cache mají omezenou kapacitu a nemají záložní paměť, až na klasifikátor v třetí úrovni. Nicméně při příchodu paketu, který způsobuje výpadek v cache, není možné čekat na výsledek klasifikátoru, neboť mezitím je potřeba zpracovat další příchozí pakety. Proto je pozornost zaměřena na velké toky, které je možné v cache udržet po celou dobu jejich existence bez výpadků.

Obvodová realizace hashovací tabulky v FPGA dovoluje využít jakoukoliv správu paměti, která je založena na definici správy dle kapitoly 5.2. Implementace hashovací tabulky využívá velké propusnosti paměti na čipu. V FPGA od firmy Xilinx řady Virtex je paměť rozdělena do malých modulů nazvaných BlockRAM (každý o kapacitě 2-4KB a maximální šířce slova 32 až 64 bitů dle typu FPGA). Díky tomu lze v jednom hodinovém taktu vyčíst data ze všech pamětí. Je tak možné realizovat hashovací tabulky s dlouhým řádkem. Na základě předchozích experimentů je délka řádku stanovena na 32 stavů. Architektura hashovací tabulky je znázorněna na obrázku 9.3. Pro jednoduchost jsou zakresleny pouze čtyři BlockRAM, nicméně pro implementaci řádku délky 32 je jich dle délky otisku potřeba 32 a více. BlockRAM jsou umístěny paralelně vedle sebe. Řádek hashovací tabulky zahrnuje slova se stejnou adresou ve všech BlockRAM. Všechny otisky v řádku jsou vyčteny najednou a paralelně porovnány s vypočítaným otiskem paketu (CMP). Jakmile je nalezen příslušný otisk, je stav toku doručen multiplexorem (MX) do výpočetní jednotky (PJ). Jednotka PU aktualizuje stav toku a uloží ho zpět do paměti. Do kritických míst jsou vloženy registry pro rozdělení dlouhých kombinačních cest. Správa paměti je implementována dle své definice přesouváním aktualizovaných stavů v řádku. To vyžaduje pomocné multiplexory, které přivádí na vstupy BlockRAM aktualizovaný stav toku nebo výstup z předchozí BlockRAM sloužící pro posun neaktualizovaných stavů směrem ke konci řádku.

Režie vstupních multiplexorů je zanedbatelná. Následně postačí dle správy paměti správně nastavit signály *we* povolující zápis do paměti a *mxs* ovládající vstupní multiplexory. Počet kombinací nastavení signálu odpovídá délce řádku plus jedna kombinace pro vložení nového stavu. Nastavení signálů je dle dané správy uloženo do konfigurační paměti a není tak nutné je s každým přístupem počítat. Změna správy znamená pouze přepsání této konfigurační paměti bez nutnosti jakkoliv měnit strukturu obvodu. V tabulce 9.2 je ukázka uložených kombinací signálů *we* a *mxs* pro řádek se čtyřmi stavy a správou $S = (2, (0, 0, 1, 1))$. V případě použití rozšířené správy využívající klasifikátor dle velikosti toku je nutné pro každou výstupní třídu rozšířit tabulku signálů. Výsledná paměť správu s klasifikátorem do čtyřech tříd je 4-krát větší. V případě technologie Virtex lze využít pro uložení LUT tabulky.

Tabulka 9.2: Nastavení signálů pro přesun stavů v řádku dle správy $S = (2, (0, 0, 1, 1))$.

Pozice	we	mxs
Přesun z 0	(1,0,0,0)	(1,0,0,0)
Přesun z 1	(1,1,0,0)	(0,1,0,0)
Přesun z 2	(1,1,1,0)	(0,1,0,0)
Přesun z 3	(1,1,1,1)	(0,1,0,0)
Vložení na 2	(0,0,1,1)	(0,0,1,0)

Správu lze alternativně implementovat bez přesunu stavů. To je výhodné v okamžiku, kdy je stav toku příliš velký na přesun v jednom taktu. Pozice stavů jsou pak uloženy ve vyhrazeném vektoru pro každý řádek. Tento vektor je s každým přesunem aktualizován. Velikost vektoru je pak $m \cdot log(m)$ bitů, kde m je kapacita řádku počítaná v počtu stavů.

9.4 Hodnocení systému

Pro ohodnocení vlastností AtoZ je systém zapojen do testovacího prostředí. Toto prostředí se skládá ze dvou výkonných počítačů, které dokáží odeslat nasbíraný provoz pomocí programu *tcpreplay*. Nasbíraný provoz představuje vzorek provozu z univerzitní sítě a výzkumné instituce, v němž jsou zachyceny vztahy mezi tokem a třídou aplikace. Složení a rozsah vzorků je popsán v [41]. Vzorky provozu jsou rozděleny dle toků na oba počítače. Počítače jsou s AtoZ propojeny přes přepínač, který zajistí spojení provozu od obou počítačů a přesměrování směrem k testovaném systému. Tím je zajištěna možnost plně zatížit jednu gigabitovou linku vedoucí k AtoZ. Na této lince je zároveň zřízen pasivní odposlech, který dovoluje sledovat chování AtoZ, například jeho zpoždění a ztráty paketů.

Systém AtoZ se skládá z karty NetFPGA a hostitelského počítače s konfigurací Intel Quad Core CPU (2.40 GHz) s 4 GB operační paměti a OS CentOS Linux 5.0 (jádro 2.6.18). Část architektury AtoZ je popsána v jazyce Verilog a přeložena pro Virtex-II-Pro-50 na kartě NetFPGA. Po syntéze nástrojem XST je dosaženo pracovní frekvence 125 MHz. Použité konfigurace HC a CT jsou uvedeny v tabulce 9.3. Zabrané zdroje FPGA ukazují, že přidané moduly zabírají do 35% zdrojů čipu. Pro realizaci paměti HC je místo BlockRAM použito LUT tabulek. To se projeví ve větší spotřebě zdrojů (*Slice*) implementujících logické funkce.

Simulací hlavní datové cesty je zjištěna zpoždění modulů nacházajících se v hlavní datové cestě. Nově vzniklé HC, CT a aplikačně-specifický modul pracují v cut-through[1] módu.

[1]Po přijetí záhlaví paketu je paket průběžně přeposílán dále.

Tabulka 9.3: Konfigurace HC a CT pro NetFPGA (Zabrané zdroje jsou výstupem překladového nástroje XST).

Parametry	HC	CT
Počet stavů	2048	32768
Otisk	40	36
Délka hash $(h + b)$	48	48
Počet Bram	0% (z 232)	28% (z 232)
Počet *Slice*	24% (z 23616)	11% (z 23616)
Pravděpodobnost kolize p_k	7.5×10^{-9}	1.9×10^{-6}

Tabulka 9.4: Zpoždění jednotlivých modulů v datové cestě.

Modul	Zpoždění	Mód	šířka sběrnice
Eth. vstup	608–11952 ns	store&forward	8 bitů
HC	224 ns	cut-through	64 bitů
CT	208 ns	cut-through	64 bitů
Ap.-sprec.	16 ns	cut-through	64 bitů
Výstupní fronty	72–1496 ns	store&forward	64 bitů
Eth. výstup	16 ns	cut-through	8 bitů

Moduly třetích stran zajišťující příjem a odesílání paketů pracují v módu store&forward[2]. Všechny pakety prochází všemi moduly bez ohledu na to, zda jsou označeny nebo nikoliv. Zpoždění modulů je zachyceno v tabulce 9.4. Doba zpoždění u modulů store&forward se liší v závislosti na délce paketu (pakety od 64 B do 1500 B). Celkové zpoždění paketu je odhadováno na 1,1 µs až 14 µs v závislosti na velikosti paketu.

Propusnost AtoZ je závislá pouze na modulu s nejdelší dobou pro zpracování jednoho paketu. Tímto modulem je HC. Úzké hrdlo je způsobeno dvěma přístupy do hash tabulky, kdy dokončení jednoho přístupu včetně vykonání správy zabere 5 taktů. Další takty jsou spotřebovány na režii spojenou s extrakcí a výpočtem hash, přestože je tato činnost zřetězena. Celkem všechny operace zabírají 15 taktů. Při frekvenci 125 MHz je možné zpracovat pakety přicházející každých 120 ns. To odpovídá propusnosti 8 milionů paketů za sekundu, což je dostatečné pro zpracování všech čtyř plně vytížených gigabitových rozhraní (max. 6 milionů paketů za sekundu).

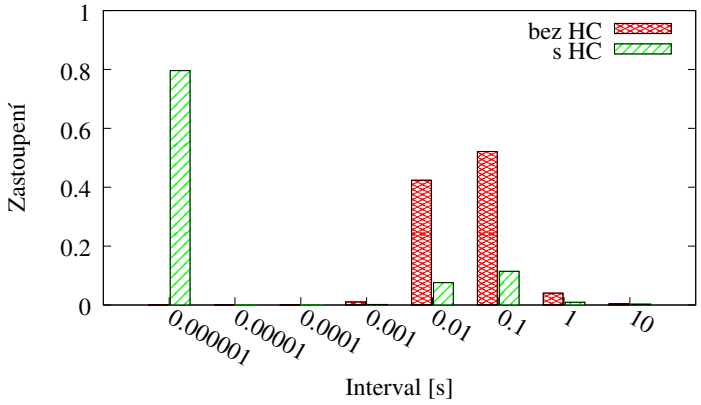
AtoZ je v realném testovacím prostředí porovnán s řešením založeným čistě na hostitelském počítači bez jakékoliv akcelerace. Toto řešení je založeno na platformě modulárního softwarového směrovače Click [38] s rozšířením pro rozpoznání aplikace. Click je nainstalován na stejném hostitelském počítači. Pro příjem paketů využívá síťovou kartu Intel e1000, neboť NetFPGA nemá dostatečnou propusnost, pokud je použita jako standardní síťová karta. S každým řešením je provedeno 10 nezávislých měření. Výsledky těchto měření pro různá zatížení jsou uvedeny v tabulce 9.5. Počítače se vzorky přehrávají provoz na různých rychlostech. Složení provozu odpovídá nasbíraným vzorkům (průměrná délka paketů je přibližně 1 KB).

Systém AtoZ si zachovává nízkou latenci (17 µs) bez jakékoliv ztráty paketu. Zároveň je procesor hostitelského počítače průměrně vytížen na 6%. Čistě softwarové řešení založené na

[2]Paket je nejprve celý uložen a následně odeslán.

Tabulka 9.5: Porovnání zpoždění AtoZ s neakcelerovaným řešením.

	300 Mb/s (51 Kpps)	600 Mb/s (107 Kpps)	1 Gb/s (137 Kpps)
AtoZ	13 ± 4 μs	16 ± 3 μs	17 ± 2 μs
pouze SW	304 ± 120 μs	12 ± 25 ms	-



Obrázek 9.4: Interval od prvního paketu toku po správné označení toku.

Click je schopno zpracovat provoz kolem 300 Mb/s bez ztráty paketu, zároveň je latence již o řád vyšší než u AtoZ. Při 600 Mb/s je na hostitelským počítačem zahazováno 4% paketů a zatížení procesoru dosahuje 100%. To způsobí nárůst zpoždění na desítky milisekund. Při dalším zvyšení zátěže je ztrátovost tak vysoká, že zpoždění již nelze spolehlivě měřit.

Další experimenty jsou provedeny s cílem porovnat vlastnosti AtoZ z pohledu vytížení jednotlivých částí systému a zjistit přesnost klasifikace. Během těchto experimentů je provoz přehrán na své původní rychlosti. Host cache je schopna vyhledat stav toku pro více než 70% toků pro obě datové sady. Významně tak snižuje počet stavů nutných pro uchování v CT a zároveň množství duplikovaných paketů odesílaných do modulu KA. HC rovněž dovoluje rozpoznat aplikaci již v prvním paketu nově příchozího toku díky stavům založených na trojici IP adresy, portu a protokolu. Histogram na obrázku 9.4 ukazuje rozložení rychlosti rozpoznání s využitím HC a bez HC. Využití HC výrazně snižuje interval mezi prvním paketem toku a správnou klasifikací toku. U 70% je tento interval menší než 10 µs. Díky použití CT je výrazně omezeno množství provozu duplikované do hostitelského počítače. V provedených experimentech množství provozu zpracované v KA odpovídalo 1‰.

Použití předem označených datových sad dovoluje ohodnotit přesnost klasifikace. Z pohledu toků dosahuje AtoZ 99,6% přesnosti značení, z pohledu paketů 99,4% přesnosti.

Kapitola 10

Závěr

Tato práce se věnovala optimalizaci sledování síťových toků z pohledu jejich správy v cache toků. Celá práce byla rozdělena do několika částí. První teoretická část popsala současné správy a shrnula vlastnosti síťového provozu včetně použitých datových sad pro následující experimenty. Následující část byla zaměřena na optimalizaci správy cache toků pro provoz dané linky. Za tímto účelem bylo navrženo použití genetického algoritmu a jeho přizpůsobení pro daný problém. GA byl schopen vyvinout správu cache, která podává lepší výsledky na testovaných sadách než ostatní správy. Přestože pro síťový provoz platí paradigma o nízkém počtu toků odpovědných za většinu provozu, správa paměti nesmí být plně zaměřena na tyto toky, pokud má dosáhnout snížení celkového počtu výpadků stavů z cache. Stavy velkých toků se i při špatné správě způsobující jejich výpadky vyskytují v cache díky vysokému počtu příchozích paketů. Pro snížení celkového počtu výpadků je pak výhodnější zaměřit se na méně intenzivní toky. Takové toky jsou k výpadkům daleko náchylnější a rozhodují o úspěšnosti správy. Toto pozorování bylo možné odvodit ze struktury aktualizačního vektoru nalezené správy GARP. Narozdíl od ostatních správ GARP neudržuje nejaktivnější stavy toků na začátku spravovaného seznamu, ale uprostřed. Stavy toků, které se vyskytnou na konci seznamu a obdrží paket, jsou přesunuty na začátek seznamu. Tím je zvýšena šance, že při příchodu dalšího paketu budou stále v cache.

V případě, že je nutné ušetřit záložní paměť nebo není možné rychle se dotázat výpočetního prvku na stav toku, pak se vyplatí využít správu, která se zaměří na stavy velkých toků. Práce ukázala, že se stejně velkou cache, jaká je využita pro cache v běžném případě, je možné udržet většinu velkých toků bez jediného výpadku. K tomu nejlépe slouží vyvinutá správa GARP-V2. První návrh tohoto přístupu byl autorem této práce popsán v příspěvku [61] a dále rozpracován v příspěvku [63]. Pokud je správa rozšířena o klasifikátor odhadující velikost toku na základě příznaků z paketu, pak správa dosahuje na vybraných sadách výsledků shodných s ideální správou. Navíc během období záplav nových toků, které se vyskytují na síti v době útoků, se GARP-V2 ukázala jako velmi robustní. Je schopna držet výpadky velkých toků na nízké hodnotě a zajistit tak dostupnost stavů pro většinu provozu. Využití GARP-V2 jako robustní správy cache síťových toků bylo autorem publikováno v [62].

I přes optimalizaci správy na danou datovou sadu je mezi nalezenou a ideální správou stále rozdíl. Proto byly hledány možnosti nápovědy správy na základě příznaků z paketů. Byl sestaven klasifikátor a zkombinován se správou paměti, což dovolilo reálnou správu dále přiblížit ideálnímu řešení.

Navrženou správu cache lze realizovat v systémech sledujících stavy toků na vysokých rychlostech. Detaily a rozšíření obvodové realizace v FPGA byly publikovány v příspěv-

ku [64]. Kromě systému pro organizaci síťového provozu [14] popsaného v této práci, lze správu využít v sondách pro sledování síťových toků popsaných v publikacích [65, 67] pro sběr a generování informací o tocích v podobě NetFlow nebo IPFIX. Sledování statistik využití cache pak může vést i k detekcím útoků, jak bylo autorem navrženo v [7]. Rozšíření správy o nápovědu pomocí klasifikátoru příznaků bylo inspirováno autorovou dřívější prací na téma detekce nevyžádané pošty na základě sledování charakteristik paketů publikovanou v [66].

10.1 Přínosy práce

V rámci této práce byl navržen GA pro vývoj správy cache toků. Tento přístup dovoluje vyvinout správu přizpůsobenou daným parametrům cache a charakteristikám provozu na dané lince.

Byl zkoumán problém nalezení správy snižující celkový počet výpadků stavů z cache. Vyvinutá správa GARP byla porovnána s ostatními správami. Výsledky ukazují, že vyvinutá správa dosáhla nejnižší pravděpodobnosti výpadků z testovaných správ založených na odvození následujícího přístupu na základě sledování předchozích přístupů.

Dále byla GA vyvinuta správa snižující počet výpadků stavů velkých toků. Vyvinutá správa GARP-V2 dosáhla nižšího počtu výpadků u velkých toků v porovnání s ostatními správami.

Práce vyhodnotila chování správ cache při síťových útocích záplavou toků. Výsledky ukázaly, že GARP-V2 je svým chováním nejblíže optimálnímu řešení.

Tato práce rovněž navrhla postup pro využití informace ze záhlaví paketů pro dosažení lepších výsledků správy cache toků. Tento postup byl experimentálně ověřen a výsledky ukázaly, že bylo dosaženo nižší pravděpodobnosti výpadků při správě cache toků i správě cache velkých toků oproti správě bez rozšíření.

V rámci práce byl navržen, implementován a zhodnocen reálný systém založený na předchozích výsledcích. Tento systém řešil úlohu zpracování (traffic-shapping) síťového provozu na platformě složené ze síťové karty s FPGA a hostitelského počítače.

10.2 Budoucí práce

Práci lze dále rozšiřovat v několika možných směrech. Prvním z nich je využití navrženého přístupu na vývoj specifických správ cache dle dané aplikace. Nabízí se například možnost využít správu na cache směrovacích tabulek. Tyto cache pracují pouze s prefixem dané IP adresy, a proto může být charakteristika dotazů do cache jiná než u klasické cache toků.

Výzkum správy cache toků může rovněž dále rozšiřovat správu o využívání příznaků z paketů. Pokud zvážíme, že by bylo možné do stavu o toku přidat dodatečnou informaci a uchovat ji dle potřeby, pak se značně zvýší možnosti správy cache. Historie klasifikace se v navržném přístupu uchovává pouze v podobě pozice stavu v řádku. Uchování historie o příznacích a klasifikaci na jemnější úrovni by tak poskytlo daleko více informací, které by bylo možné využít kdykoliv během přítomnosti stavu toku v cache.

Dalším možným směrem výzkumu správy cache je kombinace filtrovacích mechanismů a samotné správy. Vhodnou kombinací by mělo být možné ještě lépe identifikovat a uchovat velké toky. Kombinace filtru a správy cache by rovněž mohla pomoci i při snižování pravděpodobnosti výpadků všech toků, pokud by se podařilo odfiltrovat toky s velmi malým počtem paketů (typicky s jedním paketem) a zároveň neovlivnit ostatní toky.

Rovněž další zvyšování odolnosti správy cache toků vůči síťovým útokům může být velmi zajímavé téma výzkumu. Pokud by se podařilo tyto útoky spolehlivě detekovat, bylo by možné navrhnout přepínání například dvou typů správ, jedna by sloužila pro dosažení efektivních výsledků za běžného provozu a druhá by byla nasazena v době útoku. Pokud by se navíc podařilo dostatečně rychle identifikovat toky náležící samotnému útoku, pak by bylo možné navrhnout efektivní filtrovací techniky, které by zabránily vytvoření škodlivého stavu toku v cache.

Kapitola 11

Příloha

11.1 Výpis posloupností

Výpis všech 35 neklesajících posloupností vektoru V řádek délky 4:

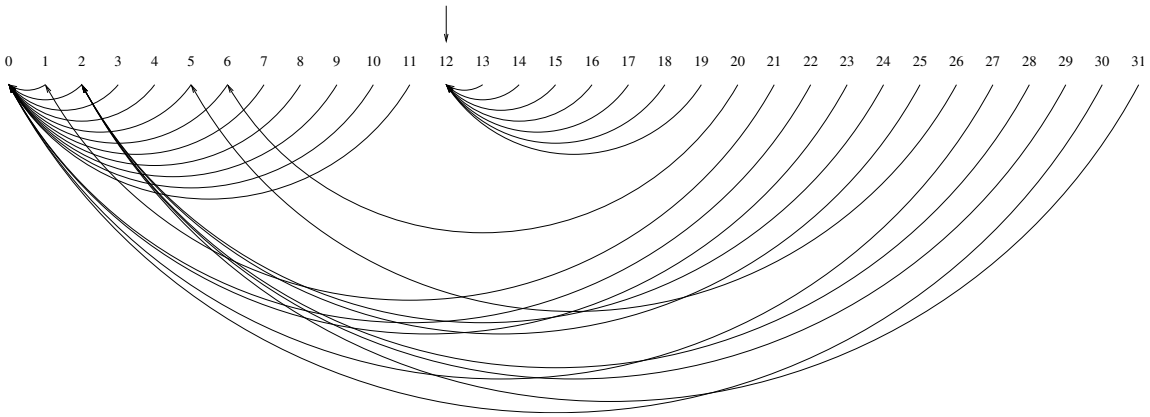
$$
\begin{gathered}
0,0,0,0 \\
0,0,0,1 \\
0,0,0,2 \\
0,0,0,3 \\
0,0,1,1 \\
0,0,1,2 \\
0,0,1,3 \\
0,0,2,2 \\
0,0,2,3 \\
0,0,3,3 \\
0,1,1,1 \\
0,1,1,2 \\
0,1,1,3 \\
0,1,2,2 \\
0,1,2,3 \\
0,1,3,3 \\
0,2,2,2 \\
0,2,2,3 \\
0,2,3,3 \\
0,3,3,3 \\
1,1,1,1 \\
1,1,1,2 \\
1,1,1,3
\end{gathered}
\tag{11.1}
$$

$$
\begin{array}{c}
1,1,2,2 \\
1,1,2,3 \\
1,1,3,3 \\
1,2,2,2 \\
1,2,2,3 \\
1,2,3,3 \\
1,3,3,3 \\
2,2,2,2 \\
2,2,2,3 \\
2,2,3,3 \\
2,3,3,3 \\
3,3,3,3
\end{array}
$$

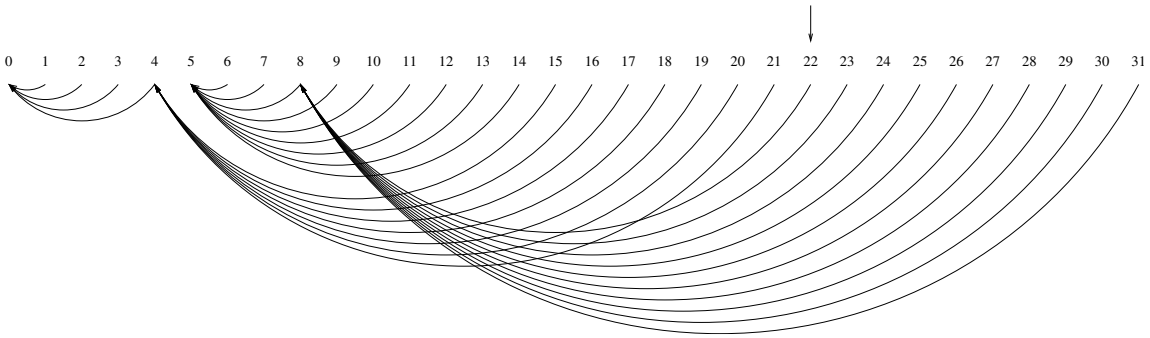
Výpis všech 14 neklesajících posloupností vektoru V řádek délky 4, spňující první 2 optimalizace genetického algoritmu:

$$
\begin{array}{c}
0,0,0,0 \\
0,0,0,1 \\
0,0,0,2 \\
0,0,0,3 \\
0,0,1,1 \\
0,0,1,2 \\
0,0,1,3 \\
0,0,2,2 \\
0,0,2,3 \\
0,1,1,1 \\
0,1,1,2 \\
0,1,1,3 \\
0,1,2,2 \\
0,1,2,3
\end{array}
\tag{11.2}
$$

11.2 Graficky zobrazené správy cache



Obrázek 11.1: Grafické zobrazení správy paměti nalezené pomocí GA při použití operátoru mutace O_{mut-3} optimalizace Opt_3 a proměnné pravděpodobnosti mutace $p_{mut}(v)$, $S = (12, (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 12, 12, 12, 12, 12, 12, 12, 6, 1, 0, 0, 2, 2, 5, 0, 2, 2, 0, 2))$.



Obrázek 11.2: Grafické zobrazení správy paměti nalezené pomocí GA na specifickém vzorku dat, $S = (22, (0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8))$.

11.3 Výsledky hledaní bodu vložení pro SLRU

Tabulka 11.1: Úspěšnost správy SLRU s různým nastavením vstupního bodu hodnocena na *Mawi-2010/04/14-14:00*.

s	exp. toků	% podíl vůči real. tokům
8	1861167	149,3
9	1859211	149,1
10	1857384	149,1
11	1856222	148,9
12	1855327	148,8
13	1855068	148,8
14	1855234	148,8
15	1856031	148,9
16	1857282	149,0

11.4 Nastavení klasifikátoru vzdáleností

weka.classifiers.trees.J48 -C 0.25 -M 2000

```
NAME
weka.classifiers.trees.J48
OPTIONS
binarySplits=FALSE        -- Whether to use binary splits on nominal
                             attributes when building the trees.
confidenceFactor=0,25     -- The confidence factor used for pruning
                             (smaller values incur more pruning).
debug=FALSE               -- If set to true, classifier may output
                             additional info to the console.
minNumObj=2000            -- The minimum number of instances per leaf.
numFolds=3                -- Determines the amount of data used
                             for reduced-error pruning.
                             One fold is used for pruning, the rest
                             for growing the tree.
reducedErrorPruning=FALSE -- Whether reduced-error pruning is used
                             instead of C.4.5 pruning.
saveInstanceData=FALSE    -- Whether to save the training data
                             for visualization.
seed=1                    -- The seed used for randomizing the data
                             when reduced-error pruning is used.
subtreeRaising=True       -- Whether to consider the subtree raising
                             operation when pruning.
unpruned=FALSE            -- Whether pruning is performed.
useLaplace=FALSE          -- Whether counts at leaves are smoothed
                             based on Laplace.
```

Výstup klasifikátoru je na následující straně.

Ukázka klasifikátoru vzdáleností:

```
iplength <= 12
|   tcpflags <= 16
|   |   tcpflags <= 2: stredni
|   |   tcpflags > 2
|   |       tcpflags <= 4: nekonecno
|   |       tcpflags > 4
|   |           tcpwindow <= 328: kratke
|   |           tcpwindow > 328
|   |               tcpwindow <= 654
|   |               |   tcpwindow <= 628: kratke
|   |               |   tcpwindow > 628
|   |               |   |   tcpwindow <= 645: kratke
|   |               |   |   tcpwindow > 645: stredni
|   |               tcpwindow > 654: kratke
|   tcpflags > 16
|       tcpflags <= 21
|       |   tcpwindow <= 28: nekonecno
|       |   tcpwindow > 28
|       |       tcpflags <= 17
|       |       |   tcpwindow <= 68: stredni
|       |       |   tcpwindow > 68: nekonecno
|       |       tcpflags > 17: stredni
|       tcpflags > 21
|           tcpwindow <= 57: stredni
|           tcpwindow > 57
|               tcpwindow <= 71
|               |   iplength <= 1: stredni
|               |   iplength > 1: kratke
|               tcpwindow > 71
|                   tcpwindow <= 642
|                   |   iplength <= 10: stredni
|                   |   iplength > 10: kratke
|                   tcpwindow > 642
|                       iplength <= 1
|                       |   tcpwindow <= 652: kratke
|                       |   tcpwindow > 652: stredni
|                       iplength > 1: stredni
iplength > 12: kratke
```

Ukázka klasifikátoru velkých toků (prvních několik uzlů z celkových 93):

```
iplen <= 1084
|   tcp_flags <= 4
|   |   iplen <= 415: L4
|   |   iplen > 415
|   |       iplen <= 593
|   |       |   iplen <= 511: L1
|   |       |   iplen > 511: L4
|   |       iplen > 593: L1
|   tcp_flags > 4
|   |   tcp_flags <= 16
|   |   |   iplen <= 74
|   |   |   |   iplen <= 46
|   |   |   |   |   tcp_window <= 8394: L4
|   |   |   |   |   tcp_window > 8394
|   |   |   |   |       tcp_window <= 65534
|   |   |   |   |       |   tcp_window <= 64512
|   |   |   |   |       |   |   tcp_window <= 17519
|   |   |   |   |       |   |   |   tcp_window <= 16905: L3
|   |   |   |   |       |   |   |   tcp_window > 16905: L4
|   |   |   |   |       |   |   tcp_window > 17519
|   |   |   |   |       |   |       tcp_window <= 64076
|   |   |   |   |       |   |       |   tcp_window <= 50575
|   |   |   |   |       |   |       |   |   tcp_window <= 17533: L3
|   |   |   |   |       |   |       |   |   tcp_window > 17533
|   |   |   |   |       |   |       |   |       tcp_window <= 25075: L4
|   |   |   |   |       |   |       |   |       tcp_window > 25075: L3
|   |   |   |   |       |   |       |   tcp_window > 50575: L2
|   |   |   |   |       |   |       tcp_window > 64076: L3
|   |   |   |   |       |   tcp_window > 64512: L4
|   |   |   |   |       tcp_window > 65534: L3
|   |   |   |   iplen > 46
|   |   |   |   |   tcp_window <= 6457
|   |   |   |   |   |   tcp_window <= 5805
|   |   |   |   |   |   |   tcp_window <= 550: L3
|   |   |   |   |   |   |   tcp_window > 550
|   |   |   |   |   |   |       tcp_window <= 1024: L2
|   |   |   |   |   |   |       tcp_window > 1024
|   |   |   |   |   |   |           tcp_window <= 2184: L3
|   |   |   |   |   |   |           tcp_window > 2184: L2
|   |   |   |   |   |   tcp_window > 5805: L4
|   |   |   |   |   tcp_window > 6457
|   |   |   |   |   |   tcp_window <= 6607: L2
.
.
.
```

11.5 Vyvinuté správy cache

Tabulka 11.2: Správy GARP pro různé datové sady.

Sada	s	V																																				
Mawi-2010/04/14-14:00	10	0	0	0	0	0	0	2	2	8	8	8	8	8	8	8	8	8	8	8	8	8	8	5	5	5	1	1	3	3	3	3	3	3				
Snjc-2009/07/17-13:00	13	0	0	1	1	1	1	1	1	1	1	1	1	5	5	5	5	6	6	6	6	6	6	0	0	0	0	0	0	0	0	0	0					
Vut-2011/10/18-15:00	13	0	0	1	1	1	1	1	1	1	0	0	0	0	13	13	13	13	13	13	13	13	8	8	8	8	4	4	4	3	3	3	3					

Tabulka 11.3: Správa GARP pro různé velikosti cache pro datovou sadu *Mawi-2010/04/14-14:00*.

Vel.	s	V																																				
4K	13	0	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4	4	4	0	0	0	0	0						
8K	10	0	0	0	0	0	0	2	2	8	8	8	8	8	8	8	8	8	8	8	8	5	5	5	1	1	3	3	3	3	3	3						
16K	7	0	0	0	0	0	0	0	7	7	7	7	7	7	7	7	7	7	7	7	3	3	3	3	3	1	1	3	3	2	2	2	2					
32K	9	0	0	1	0	0	0	0	7	7	7	7	7	7	7	7	7	7	7	2	2	2	2	2	2	1	1	1	1	1	1	1	1					

Tabulka 11.4: Správy $\text{GARP}_K - \text{K}_{ideal}$ a $\text{GARP}_K - \text{K}_{real}$ vyvinuté na různých datových sadách.

	s	V																																		U			
Mawi-2010/04/14-14:00																																						
$\text{GARP}_K - \text{K}_{ideal}$	18	0	0	0	0	0	0	0	0	1	1	1	1	6	6	6	6	6	6	0	0	0	0	0	8	8	8	8	11	7	27	27	27	7	-6	10	31
$\text{GARP}_K - \text{K}_{real}$	8	0	0	0	0	1	4	5	5	5	5	5	5	5	5	5	5	5	2	2	2	2	0	0	0	1	1	1	1	1	1	1	1	0	0	15	23
Snjc-2009/07/17-13:00																																						
$\text{GARP}_K - \text{K}_{ideal}$	21	0	0	0	2	2	2	2	2	0	0	0	0	0	0	0	0	13	13	1	1	1	1	1	4	4	4	4	6	6	6	6	23	23	-4	-2	30	31
$\text{GARP}_K - \text{K}_{real}$	5	0	0	0	0	0	0	0	5	5	6	6	6	0	0	0	0	1	1	1	6	6	1	1	17	17	17	17	17	17	17	17	18	1	4	14	17
Vut-2011/10/18-15:00																																						
$\text{GARP}_K - \text{K}_{ideal}$	23	0	0	0	2	2	2	2	2	6	6	6	6	6	6	6	6	6	6	13	13	13	18	18	18	18	18	18	27	27	8	8	8	8	0	1	29	30
$\text{GARP}_K - \text{K}_{real}$	6	0	0	1	1	1	1	1	6	6	6	3	3	3	3	3	3	3	10	10	10	5	5	12	12	24	24	24	24	7	7	7	7	-1	0	14	17

Tabulka 11.5: Správy cache velkých toků GARP-V2 pro různé datové sady.

Sada	s	V																																				
Mawi-2010/04/14-14:00	18	0	0	0	2	3	4	5	6	7	8	9	10	11	12	13	13	13	13	15	15	15	16	16	16	16	18	18	18	18	22	22	22					
Snjc-2009/07/17-13:00	20	0	0	0	2	3	4	5	6	7	7	8	9	13	13	13	13	14	14	14	14	14	18	19	20	21	22	23	24	22	22	22	22					
Vut-2011/10/18-15:00	19	0	0	0	0	0	0	0	6	6	6	6	6	11	12	13	14	15	16	16	17	18	19	21	22	23	24	25	26	20	21	22	23					

Tabulka 11.6: Správy $\text{GARP-V2}_{K_V} - \text{K}_{V,ideal}$ a $\text{GARP-V2}_{K_V} - \text{K}_{V,real}$ vyvinutá na datové sadě *Mawi-2010/04/14-14:00*.

Správa	s	V																													U			
$\text{GARP-V2}_{K_V} - \text{K}_{V,ideal}$	15	0	0	0	2	3	4	5	6	7	8	9	10	11	12	9	9	9	9	14	15	15	16	16	16	16	18	19	20	21	27	27	27	0	-3	20	30
$\text{GARP-V2}_{K_V} - \text{K}_{V,real}$	19	0	0	0	2	3	4	5	6	7	7	7	7	7	9	10	11	12	13	16	16	16	16	16	21	21	21	22	25	25	25	25	25	-1	7	21	20

Tabulka 11.7: Správy GARP-V2$_{K_V}$−K$_{V,ideal}$ a GARP-V2$_{K_V}$−K$_{V,real}$ vyvinutá na datové sadě *Snjc-2009/07/17-13:00*.

Správa	s	V																														U			
GARP-V2$_{K_V}$−K$_{V,ideal}$	24	0	0	0	0	0	0	1	2	3	4	5	6	8	8	8	8	8	8	8	14	14	14	14	14	14	14	16	16	25	25	22	22	0	-6	7	11
GARP-V2$_{K_V}$−K$_{V,real}$	29	0	0	1	1	1	3	3	3	0	0	0	0	0	0	0	0	0	0	15	15	15	15	15	15	15	15	19	19	25	26	27	28	29	-4	-7	4	10

Literatura

[1] Cisco web pages, http://www.cisco.com, 2006.

[2] Big growth for the internet ahead, cisco says, dostupné online: http://gigaom.com/2008/06/16/big-growth-for-internet-to-continue-cisco-predicts/, 2008.

[3] Internet growth statistics, http://www.internetworldstats.com/emarketing.htm, 2009.

[4] The mawi archive, http://mawi.wide.ad.jp/mawi/, 2010.

[5] Erik R. Altman, Vinod K. Agarwal, and Guang R. Gao. A novel methodology using genetic algorithms for the design of caches and cache replacement policy. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 392–399, San Francisco, CA, USA, 1993.

[6] Martin F. Arlitt and Carey L. Williamson. Trace-driven simulation of document caching strategies for internet web servers. *Simulation Journal*, 68:23–33, 1996.

[7] Václav Bartoš and Martin Žádník. Network anomaly detection: comparison and real-time issues. In *Proceedings of the 6th IFIP WG 6.6 international autonomous infrastructure, management, and security conference on Dependable Networks and Services*, AIMS'12, pages 118–121, Berlin, Heidelberg, 2012.

[8] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5:78–101, 1966.

[9] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Pop-level and access-link-level traffic dynamics in a tier-1. In *In ACM Sigcomm Internet Measurement Workshop*, pages 39–54, 2001.

[10] N. Brownlee. Understanding internet traffic streams: Dragonflies and tortoises. *IEEE Communications Magazine*, 40:110–117, 2002.

[11] R. Caceres. Measurements of wide-area internet traffic. Technical report, University of California, Berkeley, 1989.

[12] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area tcp/ip conversations. In *Proceedings of ACM SIGCOMM '91*, pages 101–112, 1991.

[13] A.M. Campoy, I. Puaut, A.P. Ivars, and J.V.B. Mataix. Cache contents selection for statically-locked instruction caches: an algorithm comparison. In *Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 49 – 56, 2005.

[14] Marco Canini, Wei Li, Martin Žádník, and Andrew W. Moore. Experience with high-speed automated application-identification for network-management. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '09, pages 209–218, New York, NY, USA, 2009.

[15] M. Casado, M.J. Freedman, J. Pettit, Jianying Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking Enterprise Network Control. *IEEE/ACM Trans. Networking*, 17(4):1270 –1283, 2009.

[16] Jongmoo Choi, Sam H. Noh, Sang Lyul Min, and Yookun Cho. An implementation study of a detection-based adaptive block replacement scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 1999.

[17] Marek Chrobak, Gerhard J. Woeginger, Kazuhisa Makino, and Haifeng Xu. Caching is hard: even in the fault model. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'10, pages 195–206, Berlin, Heidelberg, 2010.

[18] K. C. Claffy, H. Braun, and G. C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications*, 13:1481–1494, 1995.

[19] B. Claise. Cisco systems netflow services export version 9. RFC(Informational) 3954, Internet Engineering Task Force, 2004.

[20] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), 2008.

[21] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.

[22] John Dilley, Martin Arlitt, and Stephan Perret. Enhancement and validation of squid's cache replacement policy. Technical report.

[23] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 883–892, New York, NY, USA, 2007.

[24] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, 2004.

[25] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[26] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Global Telecommunications Conference*, Dec 1999.

[27] Gideon Glass and Pei Cao. Adaptive page replacement based on memory reference behavior. In *Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '97, pages 115–126, New York, NY, USA, 1997.

[28] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA, 1st edition, 1989.

[29] Danhua Guo, Guangdeng Liao, Laxmi N. Bhuyan, Bin Liu, and Jianxun Jason Ding. A scalable multithreaded l7-filter design for multi-core servers. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 60–68, New York, NY, USA, 2008.

[30] Avinatan Hassidim. Cache replacement policies for multicore processors. In Andrew Chi-Chih Yao, editor, *ICS*, pages 501–509, 2010.

[31] Wen-Chi Hou and Suli Wang. Size-adjusted sliding window lfu - a new web caching scheme. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications*, DEXA '01, pages 567–576, London, UK, 2001.

[32] R. Jain and S. A. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4:986–995, 1986.

[33] Song Jiang and Xiaodong Zhang. Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *SIGMETRICS Perform. Eval. Rev.*, 30:31–42, 2002.

[34] Song Jiang and Xiaodong Zhang. Making LRU Friendly to Weak Locality Workloads: A Novel Replacement Algorithm to Improve Buffer Cache Performance. *IEEE Trans. Comput.*, 54(8):939–952, 2005.

[35] Theodore Johnson and Dennis Shasha. 2q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 439–450, San Francisco, CA, USA, 1994.

[36] Paul Kaufmann, Christian Plessl, and Marco Platzner. Evocaches: Application-specific adaptation of cache mappings. In *Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems*, AHS '09, pages 11–18, Washington, DC, USA, 2009.

[37] Jong Min Kim, Jongmoo Choi, Jesung Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 9–9, Berkeley, CA, USA, 2000.

[38] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[39] K. Lan and J. Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46 – 62, 2006.

[40] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.

[41] Wei Li, Marco Canini, Andrew W. Moore, and Raffaele Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Comput. Netw.*, 53(6):790–809, 2009.

[42] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. New York, NY, USA, 1990.

[43] Nimrod Megiddo and Dharmendra S. Modha. Outperforming lru with an adaptive replacement cache algorithm. *Computer*, 37(4):58–65, 2004.

[44] Jeffrey C. Mogul. Tcp offload is a dumb idea whose time has come. In *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*, pages 5–5, Berkeley, CA, USA, 2003.

[45] R. Nossenson and H. Attiya. The distribution of file transmission duration in the web: Research articles. *Int. J. Commun. Syst.*, 17(5):407–419, 2004.

[46] A. Odlyzko. Internet growth: Myth and reality, use and abuse. *Journal of Computer Resource Management*, 102:23–27, 2001.

[47] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The lru-k page replacement algorithm for database disk buffering. *SIGMOD Rec.*, 22:297–306, 1993.

[48] K. Park, G. T. Kim, and M. E. Crovella. The protocol stack and its modulation effect on self-similar traffic. In Kihong Park and Walter Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*. New York, 1999.

[49] Kihong Park, Gitae Kim, and Mark Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the 1996 International Conference on Network Protocols (ICNP '96)*, ICNP '96, pages 171–, Washington, DC, USA, 1996.

[50] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.

[51] J. Postel. Internet protocol. RFC 791, Internet Engineering Task Force, 1981.

[52] J. Ross Quinlan. *C4.5: programs for machine learning*. San Francisco, CA, USA, 1993.

[53] Mike Reddy and Graham P. Fletcher. Intelligent web caching using document life histories: A comparison with existing cache management techniques. In *In 3rd International WWW Caching Workshop*, pages 35–50, 1998.

[54] K. C. Claffy S. McCreary. Trends in wide area ip traffic patterns a view from ames internet exchange, 2000.

[55] S. Sarvotham, R. Riedi, and R. Baraniuk. Connection-level analysis and modeling of network traffic. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 99–103, New York, NY, USA, 2001.

[56] R. D. Smith. The Dynamics of Internet Traffic: Self-Similarity, Self-Organization, and Complex Phenomena. *ArXiv e-prints*, 2008.

[57] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.

[58] Athena Vakali. Lru-based algorithms for web cache replacement. In *Proceedings of the First International Conference on Electronic Commerce and Web Technologies*, EC-WEB '00, pages 409–418, London, UK, 2000.

[59] U. Vallamsetty, P. Mohapatra, R. Iyer, and K. Kant. Improving cache performance of network intensive workloads. In *Proceedings of the International Conference on Parallel Processing*, pages 87 –94, Washington, DC, USA, 2001.

[60] Zdeněk Vašíček, Martin Žádník, Lukáš Sekanina, and Jiří Tobola. On evolutionary synthesis of linear transforms in fpga. In *Evolvable Systems: From Biology to Hardware*, LNCS 5216, pages 141–152, 2008.

[61] Martin Žádník and Marco Canini. Evolution of cache replacement policies to track heavy-hitter flows. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pages 31:1–31:2, New York, NY, USA, 2010.

[62] Martin Žádník and Marco Canini. Evaluation and design of cache replacement policies under flooding attacks. In *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1292 –1297, 2011.

[63] Martin Žádník and Marco Canini. Evolution of cache replacement policies to track heavy-hitter flows. In Neil Spring and George Riley, editors, *Passive and Active Measurement*, volume 6579 of *Lecture Notes in Computer Science*, pages 21–31. 2011. 10.1007/978-3-642-19260-93.

[64] Martin Žádník, Marco Canini, Andrew W. Moore, David J. Miller, and Wei Li. Tracking elephant flows in internet backbone traffic with an fpga-based cache. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 640 –644, 2009.

[65] Martin Žádník, Jan Kořenek, Ondřej Lengál, and Petr Kobierský. Network probe for flexible flow monitoring. In *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 213–218, 2008.

[66] Martin Žádník and Zbyněk Michlovský. Is spam visible in flow-level statistics? In *Networking Studies III, Selected Technical Reports*, pages 67–78, Prague, CZ, 2009.

[67] Martin Žádník, Tomáš Pečenka, and Jan Kořenek. Netflow probe intended for high-speed networks. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL05)*, pages 695–698, Tampere, FI, 2005.

[68] Colby Walsworth, Emile Aben, kc claffy, and Dan Andersen. The caida anonymized 2009 internet traces, http://www.caida.org/data/passive/passive_2009_dataset.xml, 2009.

[69] D J Watts and S H Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–2, 1998.

[70] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. Netw.*, 5(1):71–86, 1997.

[71] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA, 2005.

[72] Junbiao Zhang, Rauf Izmailov, Daniel Reininger, Maximilian Ott, and Nec U. S. A. Web caching framework: Analytical models and beyond. In *Proceedings of the 1999 IEEE Workshop on Internet Applications*, pages 132–140, Washington, DC, USA, 1999.

[73] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 309–322, New York, NY, USA, 2002.